Modelling, Formal Refinement and Partitioning Strategies for a Small Aircraft Fuel Pump System in Hybrid Event-B

Richard Banach[a]

[a]*School of Computer Science, University of Manchester, Oxford Road, Manchester, M13 9PL, U.K.*

Abstract

A case study centred on a fuel supply system for a small aircraft is presented in Hybrid Event-B, an extension of conventional Event-B that allows for the modelling and verification of hybrid and cyber-physical systems exhibiting nontrivial continuous behaviour. In contrast to many such case studies, which concentrate predominantly on timing issues, the focus in the present work is on nontrivial physical behaviour, and on the effect that this has on various refinement and partition strategies. More liberal proof obligations are developed to add flexibility to the decomposition process.

1. Introduction

In today's ever-increasing interaction between digital devices and the physical world, formalisms are needed to express the more complex behaviours that this allows. Furthermore, these days, it is no longer sufficient to focus on isolated systems, as it is more and more the case that families of such systems are coupled together using communication networks, and can thus influence each others' working. Today, the concept of *Cyber-Physical Systems* [29, 44, 48, 1, 20] has risen to prominence. These new kinds of system throw up novel challenges in terms of design technique, and it is proving more and more difficult to ignore the continuous characteristics in their behaviours, especially if designers want to engineer close to optimal values of system parameters.

The B-Method has long been well extablished as a methodology for modelling and verification of discrete event systems. The standard reference for the classical B-Method is [2]. The classical method emphasised accumulation of submodels into a reference abstract model, to be followed by relatively monolithic refinement of this towards implementation, ending in machine generated compilable and runnable source code (in a language such as C, for example). The classical B-Method and its Atelier B toolkit is by now well established as the certified development engine behind many automated urban rail systems [35].

In the last decade or so, the B-Method evolved into a more flexible modelling and verification framework, Event-B [3]. In Event-B, action refinement [9, 10, 12] is the main underlying mechanism for using refinement to accumulate design detail. The Event-B approach, and its supporting tool Rodin [4, 42] has proved to be popular in the model based development world [47].

However, despite this, the purely discrete event foundation of Event-B makes it poorly adapted to the needs of continuously evolving behaviour such as that found in cyberphysical systems. Therefore, Hybrid Event-B [18, 19] has been introduced to bring continuous capabilities to the traditionally based discrete Event-B, in order to address some of the challenges referred to. Earlier applications of this formalism include [17, 16, 13, 15]. As described below, traditional discrete Event-B events serve as the

Email address: richard.banach@manchester.ac.uk (Richard Banach)

'mode events' that interleave the 'pliant events' of Hybrid Event-B. The latter express the continuously varying behaviour of a hybrid formalism that includes both kinds of event. In this manner, a rigorous link can be made between continuous and discrete update, as needed in contemporary applications.

In this paper, we present a case study based on a fuel pumping system in a small aircraft. Unlike many case studies of cyberphysical systems targetted at the verification domain, where there is an emphasis on timing considerations, there is a preponderance of focus on physical behaviour in this case study, which brings the physical modelling capabilities of Hybrid Event-B to the fore. Besides this, we explore the ramifications of different partition and refinement strategies in the given context. As we explain below, there are non-trivial consequences of different choices regarding these aspects when we have continuous state update, compared with the situation for pure isolated instantaneous state update. Exploring these topics in detail, as we do in this paper, constitutes a significant exercise in the application of the structuring tools of multi-machine Hybrid Event-B, providing valuable insight into best practice.

The main contributions of this paper are: (a) the exercising of the structuring, decomposition and refinement capabilities of multi-machine Hybrid Event-B in an application requiring significantly more involvement of pliant variables and pliant events than in previous case studies; (b) the eliciting of best practice from the experience gained; (c) the enrichment of a number of structuring and decomposition conditions from the original multi-machine Hybrid Event-B proposal [19], generating PO schemas that add to the flexibility of these mechanisms when applied in practical contexts. Ideally, all this would have been done with the help of tool support for Hybrid Event-B. However, at the time of writing the intended tool support has not yet been implemented. To mitigate this, care has been taken to keep the case study simple enough, and the description of it detailed enough, that following the paper account is not overly taxing. This also enables its verification to be reduced to simple checks whose truth follows essentially by inspection (once one is familiar with the theory), which was the approach used in the paper.

This paper arose as an extension of [14], but gives the background theory in greater depth. The rest of the paper is as follows, including an account of how it differs from [14]. Section 2 gives the background on the fuel pumping system we study, indicating some of the inherent engineering challenges. Section 3 outlines the main elements of Hybrid Event-B. An outline of how the semantics works is given, based on [18]. Section 4 extends this to survey the multi-machine theory and its principal structuring features, and outlines how the single machine semantics is bootstrapped to provide multi-machine semantics. Section 5 discusses the structural constraints in [19] that govern the decomposition process, and how they can be relaxed to yield proof obligations that enable decomposition to be done in a more flexible manner.

Section 6 gives the top level model of the fuel system in Hybrid Event-B (which is identical to the version in [14]). After this, the developments in [14] and here part company. The entire development in [14] is done monolithically, to save space, whereas here, starting from the first refinement which is also contained in this section, the development is decomposed into different components, to utilise the multi-machine features, as stated above. Inevitably, this results in a larger text, since the formal text has to not only capture the model itself, but has to contain elements that precisely capture how all the different pieces are connected together. Section 7 covers the refinement that introduces the non-trivial continuous behaviour that is of interest here, again in the multi-machine version. Section 8 considers further strategies for partitioning and refinement, introducing the concepts of partitioning in space and in time. Although [14] discusses these ideas briefly, their detailed development is exclusive to the present paper. Thus, Section 9 considers doing space partitioning followed by time partitioning, and pursues the development according to that strategy, while Section 10 considers the opposite order, and pursues the development according to the alternative strategy. Section 11 compares the two approaches and elicits general recommendations for partition and refinement that are widely applicable. Section 12 revisits the discussion of Section 5 and reviews how the ideas there are reflected in the development just done. Section 13 describes related approaches, and Section 14 concludes.

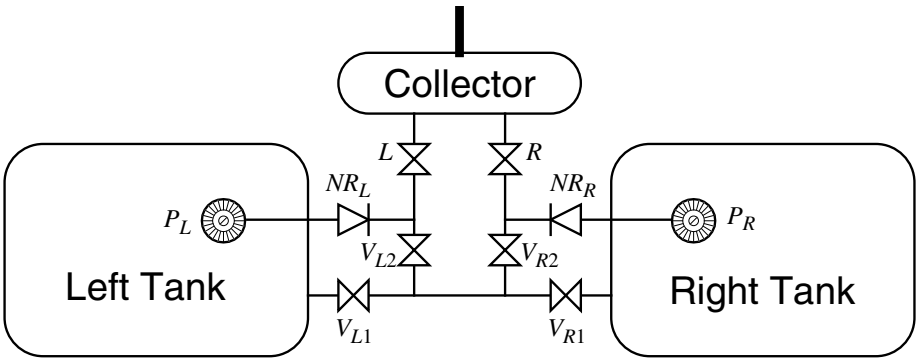

Figure 1: A schematic of a small aircraft fuel delivery system.

2. A Simplified Aircraft Fuel System

Fig. 1 outlines some elements of a simplified engine fuel delivery system for a light aircraft. The aircraft engine itself, not shown in Fig. 1, receives fuel via a high pressure pump from the relatively small Collector tank. This high pressure system is beyond the scope of our study. The collector tank in turn is fed from the main left and right fuel tanks, contained in the wings. An arrangement of pipework and valves is in place to enable fuel to move from the main tanks to the collector, and between the two main tanks. In addition to these components, there is often also a reserve tank to provide additional fuel supplies for emergency situations. This too is beyond the scope of this study. Many variations on this scheme are possible, and found in practice on various types of aircraft.

Each of the two main tanks has a low pressure pump; these are P_L and P_R in Fig. 1. The pumps have bypass mechanisms so that if the relatively low pump pressure is not sufficient to cause the flow of fuel out of the tank then the fuel is simply returned to the tank without damage to any part of the apparatus (for instance if the needed valves are not open, or if there is no more room in the fuel system downstream of the pump, or if there is a blockage in the pipe system in some inopportune place). This also protects against hydraulic hammer.[1]

Immediately beyond the pumps are non-return valves NR_L and NR_R. Beyond the non-return valves there are various pipes and valves to allow various flow arrangements as described. L and R are the (two way) valves that allow fuel to move into the collector tank from the left and right main tanks respectively. There are also further two way valves $V_{L1}, V_{L2}, V_{R1}, V_{R2}$. Two fuel gauges, G_L and G_R, inform the cockpit of the current amount of fuel in the tanks.

It is clear that if (say) all the right valves are closed, and all the left valves are open, then at least part of any fuel pumped from the left tank will return to the tank via V_{L1} and V_{L2}, depending on the relative hydraulic resistence in the various pipes, decreasing the flow into the collector tank, even though L is open. So it is important that in order to achieve a desired movement of fuel, not only must certain valves be open, but others must also be closed.

Two controls are provided within the framework we work with in this paper. The **fuel pump control** may be $OFF, BOTH, LEFT, RIGHT$. Also the **fuel rebalance control** may be $OFF, L2R, R2L$. These controls are independent, aside from the constraint that it is forbidden that when the engine is being fed by a single pump, P_L or P_R, that that same pump, P_L or P_R respectively, is simultaneously rebalancing

[1] Hydraulic hammer is the phenomenon of shock waves propagating round a hydraulic circuit following sudden movements in parts of the circuit, such as when valves are switched on or off in a high pressure circuit. Hydraulic hammer can cause severe damage to equipment if not defended against properly.

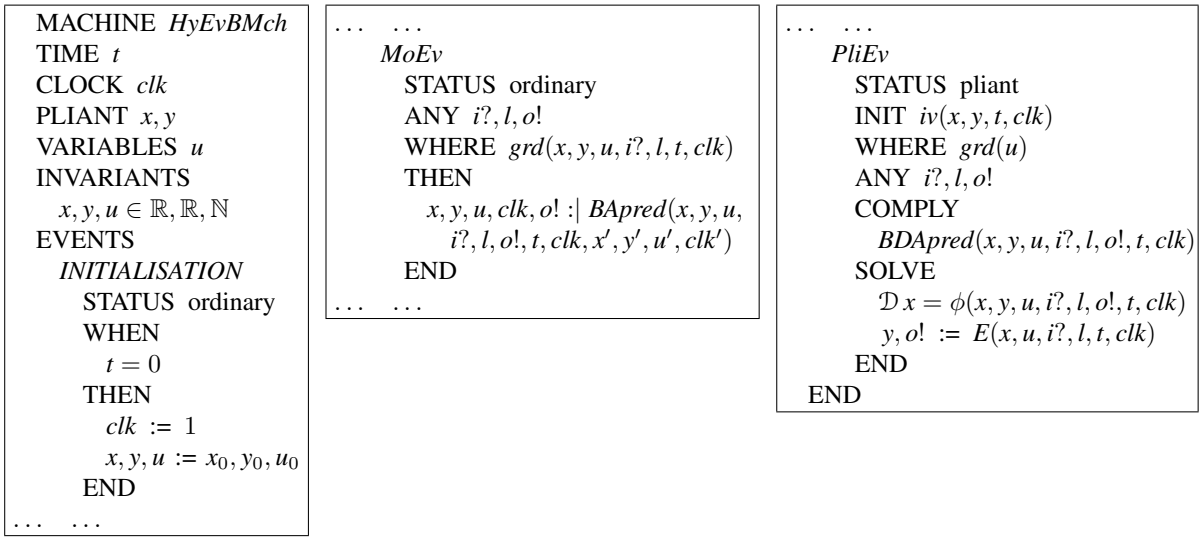

Figure 2: A schematic Hybrid Event-B machine.

fuel to the other tank.

In the framework of this paper, we treat the output of the fuel gauges as information for the pilot. This information can obviously influence the pilot's decisions on the use of the fuel pump and fuel rebalance controls, but for this paper, the gauges remain outside the control loop. In a realistic system, there will be various signals in the cockpit when the current state of the fuel system enters an undesirable regime, but we do not include such considerations in this paper.

Many details of a practical system have been omitted from the preceding account. For example, there are usually two pumps per tank, one mechanically driven from the engine for normal operation, and the other electrically driven, for engine startup, and as a fallback in case the other pump fails.

Aside from the features noted above, the fuel system of an aircraft must have a large number of additional capabilities. It must function properly, keeping the engine fed with fuel, if (even a considerable amount of) water gets into the fuel system (which must also be prevented from freezing). It must not allow an excessive amount of air into the system (which could cause engine failure) regardless of the altitude that the aircraft reaches. Along with the preceding, the fuel tanks must be properly vented to the outside air so that depletion of fuel does not cause negative relative pressure in the tanks, (which would cause potential starvation of the fuel supply to the engine). Venting notwithstanding, the fuel system must keep working properly even when the aircraft is flying upside down (if it is licensed to do so). The fuel system must prevent ignition of fuel vapour when the aircraft is hit by lightning. The list goes on. A good idea of the true complexity of the fuel supply system problem may be gained from Chapter 14 of [46].

3. An Outline of Hybrid Event-B

In this section we outline Hybrid Event-B for a single machine. We will also need to consider multiple machines, which are examined in the next two sections. Further comments on various semantic details are included in the context of the machines of our case study.

3.1. Single Hybrid Event-B Machines

In Fig. 2 we see a bare bones Hybrid Event-B machine, *HyEvBMch*. It starts with declarations of time and of a clock. In Hybrid Event-B time is a first class citizen in that all variables are functions of

time, whether explicitly or implicitly. However time is special, being read-only and never being assigned, since time cannot be controlled by any human-designed engineering process. Clocks allow a bit more flexibility, since they are assumed to increase their value at the same rate that time does (i.e. one unit per unit of time), but may be set during mode events (see below).

Variables are of two kinds. There are mode variables (like u, declared in the usual manner) which take their values in discrete sets and change their values via discontinuous assignment in mode events. There are also pliant variables (such as x, y), declared in the PLIANT clause, which take their values in topologically dense sets (normally $\mathbb{R}$) and which are allowed to change continuously; these changes are specified via pliant events (see below).

Next are the invariants. These resemble invariants in discrete Event-B, in that the types of the variables are asserted to be the (static) sets from which the variables' values *at any given moment of time* are drawn. More complex invariants are similarly predicates involving all the variables that are required to hold *at all moments of time* during a run.

Then we get to the events. The *INITIALISATION* has a guard that synchronises time with the start of any run (the WHEN clause), while all other variables are assigned their initial values in the usual way (in the THEN clause that complements the WHEN clause). As hinted above, in Hybrid Event-B, there are two kinds of event: mode events and pliant events.

Mode events are direct analogues of events in discrete Event-B. They can assign all machine variables (except time itself). In the schematic *MoEv* of Fig. 2, we see three parameters $i?, l, o!$, (an input, a local parameter, and an output respectively), and a guard *grd* which can depend on all the machine variables, and defines mode event enabledness. We also see the generic after-value assignment specified by the before-after predicate *BApred*, which can specify how the after-values of all variables (except time, inputs and locals) are to be determined. The usual abbreviations using assignment notation such as := are available.

Pliant events are new to Hybrid Event-B. They specify the continuous evolution of the pliant variables over an interval of time. The schematic pliant event *PliEv* of Fig. 2 shows the structure. There are two guards: there is *iv*, for specifying enabling conditions on the pliant variables, clocks, and time; and there is *grd*, for specifying enabling conditions on the mode variables. Their conjunction defines pliant event enabledness. The separation between the two guards is motivated by considerations connected with refinement (discussed in detail in [18]).

The body of a pliant event contains three parameters $i?, l, o!$, (once more an input, a local parameter, and an output respectively) which are functions of time, defined over the duration of the pliant event. The behaviour of the event is defined by the COMPLY and SOLVE clauses. The SOLVE clause specifies behaviour fairly directly using two specification mechanisms: direct assignments and ordinary differential equations (ODEs). For example the behaviour of pliant variable y and output variable $o!$ is given by a direct assignment to the (time dependent) value of the (vector valued) expression $y, o! := E(\ldots)$. By contrast, the behaviour of pliant variable x is given by the solution to the first order ODE $\mathcal{D}\, x = \phi(\ldots)$, where $\mathcal{D}$ indicates differentiation with respect to time. (In fact the semantics of the $y, o! := E$ case can be given in terms of the ODE $\mathcal{D}\, y, \mathcal{D}\, o! = \mathcal{D}\, E$, so that x, y and $o!$ satisfy the same regularity properties.) The COMPLY clause can be used to express any additional constraints that are required to hold during the pliant event via its before-during-and-after predicate *BDApred*. Typically, constraints on the permitted range of values for the pliant variables, and similar restrictions, can be placed here.

The COMPLY clause has another purpose. When specifying at an abstract level, we do not necessarily want to be concerned with all the details of the dynamics — it is often sufficient to require some global constraints to hold which express the needed safety properties of the machine's plaint events. (Often these are refined to more deterministic behaviour at lower levels of abstraction.) The COMPLY clauses of the relevant pliant events can house such constraints directly, leaving it to lower level refinements to add the necessary details of the dynamics.

If, from Fig. 2, we erase time, clocks, pliant variables and pliant events, we arrive at a skeleton (conventional) Event-B machine. This simple erasure process illustrates (in reverse) the way that Hybrid Event-B has been designed as a clean extension of the original Event-B framework. The only difference of note is that, now —at least according to the (conventional) way that Event-B is interpreted in the physical world— (the mode) events (left behind by the erasure) execute *lazily*, i.e. *not* at the instant they become enabled (which is, of course, the moment of execution of the previous event).

3.2. Semantics of Single Hybrid Event-B Machines

This section summarises the essentials of single machine Hybrid Event-B semantics that we need for the models of this paper. This is taken from [18], where further details and references can be found.

For a machine, such as *HyEvBMch*, the semantics is an operational semantics that constructs *system traces*. A system trace is a set of functions of time, one for each variable v declared in *HyEvBMch*, recording the value of v throughout a run of the machine. The semantics $\mathcal{S}$ of *HyEvBMch*, is the set of all system traces.

Time is modeled as an interval $\mathcal{T}$ of the reals. A run starts at some initial moment of time, t_0 say, and lasts either for a finite time, or indefinitely. So for *HyEvBMch*, $\mathcal{S}$ would consist of all system traces for clk, x, y, u, each defined over the duration of its run, which all start at $t = 0$.

Every system trace in the semantics must consist of *piecewise absolutely continuous* functions of time, with each piece being absolutely continuous on a left-closed right-open time interval such as $[t_i \ldots t_{i+1})$ where $t_i < t_{i+1}$. This is regardless of whether the piece arises from: (a) a COMPLY clause (in which case only piecewise absolutely continuous functions satisfying *BDApred* are considered); (b) an ODE with RHS which is Lipschitz continuous in the variables and measurable in time (in which case absolute continuity of the solution is guaranteed); (c) a direct assigment with RHS which is itself absolutely continuous; (d) any consistent combination of (a)-(c); (e) a mode variable's value during the interval (which remains constant except at mode transitions).

The duration of the run $\mathcal{T}$, thus breaks up into a succession of left-closed right-open subintervals: $\mathcal{T} = [t_0 \ldots t_1), [t_1 \ldots t_2), [t_2 \ldots t_3), \ldots$, in which mode transitions, effecting discontinuous updates, take place at the isolated times corresponding to the common endpoints of these subintervals t_i, and in between, the mode variables are constant and the pliant events stipulate continuous change in the pliant variables.

The operational semantics of Hybrid Event-B constructs system traces via an abstract (i.e. non-executable) algorithmic process, extending a system-trace-to-be, event execution by event execution, and replicating system-traces-to-be over available choices when choice points are encountered during an extension step. Evidently, without further control, such an approach can easily run into inconsistency. The full description in [18] contains many 'runtime checks' that pick up such inconsistencies and eliminate the corresponding system-trace-to-be from the semantics. We omit these here, firstly for brevity (since we never need them in the models below), and secondly because they can be prevented by verifying the Hybrid Event-B proof obligations for a given model during a static analysis.

The construction of system traces for a machine *M* can be summarised as follows.

[1] Let $\eta := 0$.

[2] CHOOSE an initial assignment to all variables satisfying all the invariants of *M*, thereby interpreting their values at time t_0.

[3] With the state variables having the values at t_η, CHOOSE an enabled pliant event *PliEv* and CHOOSE a simultaneous piecewise absolutely continuous solution, in a maximal left-closed, right-open interval $[t_\eta \ldots t_{\text{MAX}})$, of all the differential equations and direct assignments in the SOLVE clause of *PliEv*, using state variable values at t_η as initial values, with these initial values required to satisfy the INIT and WHERE guards of *PliEv*, and with inputs and local parameters where needed, such

that *BDApred* in the COMPLY clause of *PliEv* is also satisfied in the interval, and all the invariants of M are maintained. Use the solution to assign the values of all pliant variables (and outputs) in $[t_\eta \ldots t_{\text{MAX}})$.

[3.1] For every mode variable, extend its value at t_η to a constant function in the interval $[t_\eta \ldots t_{\text{MAX}})$.

[4] If no non-*INITIALISATION* mode event is enabled by the values of the state variables at any time in the open interval $(t_\eta \ldots t_{\text{MAX}})$ (including left-limit at t_{MAX} itself), together with a choice of values for inputs and local parameters, then TERMINATE.

[5] CHOOSE $t_{\eta+1} > t_\eta$ such that **either** $t_{\eta+1}$ is the earliest time at which a non-*INITIALISATION* mode event without inputs is enabled, **or** a non-*INITIALISATION* mode event with inputs is enabled at $t_{\eta+1}$ and there is no non-*INITIALISATION* mode event without inputs that is enabled within $(t_\eta \ldots t_{\eta+1})$.

[6] Let $\eta := \eta + 1$.

[7] CHOOSE a mode event that is enabled by the values of variables at t_η (or their left-limit values if $t_\eta = t_{\text{MAX}}$), and any needed inputs and locals, and assign to all state variables and outputs according to its *BApred*, such that all the invariants of M are satisfied, thereby (re)interpreting those variable values at t_η.

[7.1] For any other state variable without a value at t_η, interpret its value at t_η as its left-limit at t_η.

[7.2] Discard the interpretation of all state variables in the open interval $(t_\eta \ldots t_{\text{MAX}})$.

[8] Goto **[3]**.

That the above abstract procedure does not fail during the construction of system runs can be guaranteed by confirming the numerous Hybrid Event-B proof obligations (POs), discussed in detail in [18]. These can be summarised as follows.

- Initial states are well defined (feasible).
- Feasible initial states satisfy the invariants.
- Mode events which are enabled in invariant states have well defined after-states (i.e., are feasible).
- Feasible mode events reestablish the invariants.
- Pliant events which are enabled in invariant states have time-indexed families of well defined after-states that satisfy all the clauses in the event's specification, in some left-closed right-open time interval (i.e., are feasible). Optionally, the length of the interval must reach a Zeno lower bound.
- Feasible pliant event after-state families preserve the invariants at least until a preemption point.
- The after-state of any mode event disables mode events and enables some pliant event.
- The after-state family of any non-FINAL pliant event enables some mode event (the earliest time for this defining the pliant event's preemption point).

Besides the above, there are the Lipschitz and measurability properties of any ODE RHS to be checked. These follow readily for practical problems. The Zeno check is optional since it is usually impossible to verify it without solving the entire dynamics first, whereas the static checks are intended to justify avoiding doing exactly that. The FINAL designation permitted for pliant events is intended to prevent the last condition above from producing errors for events that are designed to complete a run.

We can summarise the above picture of the semantics in a more intuitive way thus:

[A] Every enabled mode event is feasible, i.e. has an after-state, and on its completion enables a pliant event (but does not enable any mode event).[2]

[2] If a mode event has an input, the semantics *assumes* that its value only arrives at a time strictly later than the previous mode event, ensuring part of [A] and [B] automatically.

[B] Every enabled pliant event is feasible, i.e. has a time-indexed family of after-states, and EITHER:

(i) During the run of the pliant event a mode event becomes enabled. It preempts the pliant event, defining its end. ORELSE

(ii) During the run of the pliant event it becomes infeasible: finite termination. ORELSE

(iii) The pliant event continues indefinitely: nontermination.

3.3. Single Hybrid Event-B Machine Refinement

Hybrid Event-B machines are developed by refinement. A concrete (refining) machine is like any other machine, with two provisos. Firstly, each concrete event must declare which abstract event it refines, unless it is a 'new' mode event — 'new' pliant events must also declare a refining abstract event. Secondly, the relationship between abstract and concrete state spaces is captured in a retrieve (or gluing) relation, also referred to as the joint invariant (supported by input and/or output relations, as needed).

A concrete machine has to obey the POs above, where 'invariants' is always interpreted to include the joint invariant with its 'dangling abstract variables' existentially quantified. Additional POs govern the refinement process itself, described in detail in [18]. Summarising, we have the following.

- If, in an invariant concrete state, a refining concrete mode event is enabled, then its abstract counterpart is enabled in a corresponding abstract state (identified via the retrieve relation).
- If, in an invariant abstract state, any abstract mode event is enabled, then in a corresponding concrete state (identified via the retrieve relation), some concrete mode event (whether refining or new) is enabled.
- If, in an invariant concrete before-state connected to a corresponding abstract before-state via the retrieve relation, a refining concrete mode event makes a transition, then there is a transition of its abstract counterpart from the abstract before-state to an after-state connected via the retrieve relation to the concrete after-state.
- If, in an invariant concrete before-state connected to a corresponding abstract state via the retrieve relation, a new concrete mode event makes a transition, then its after-state is connected via the retrieve relation to the same abstract state.
- Every transition of a concrete new mode event decreases a variant function.
- If, in an invariant concrete state, a concrete pliant event is enabled, then its abstract counterpart is enabled in a corresponding abstract state (identified via the retrieve relation).
- If, in an invariant abstract state, any abstract pliant event is enabled, then in a corresponding concrete state (identified via the retrieve relation), some concrete pliant event (whether refining or new) is enabled.
- If, in an invariant concrete before-state connected to a corresponding abstract before-state via the retrieve relation, a concrete pliant event makes a transition, then there is a transition of its abstract counterpart from the abstract before-state, such that for all times during that transition, the current abstract and concrete states are connected via the retrieve relation.

The last of these is based on the premiss that time flows at the same rate in abstract and concrete models. For theoretical convenience, we assume the sets of variables used in the two machines are disjoint. But for refinements which just add variables and behaviour to an existing model (as we typically do in this paper), we include the abstract variables among the concrete variables and presume the retrieve relation to be the natural projection from concrete to abstract.

With these POs verified, it becomes possible to prove that every concrete run has a simulating abstract run with corresponding transitions matching at suitable times during the run [18].

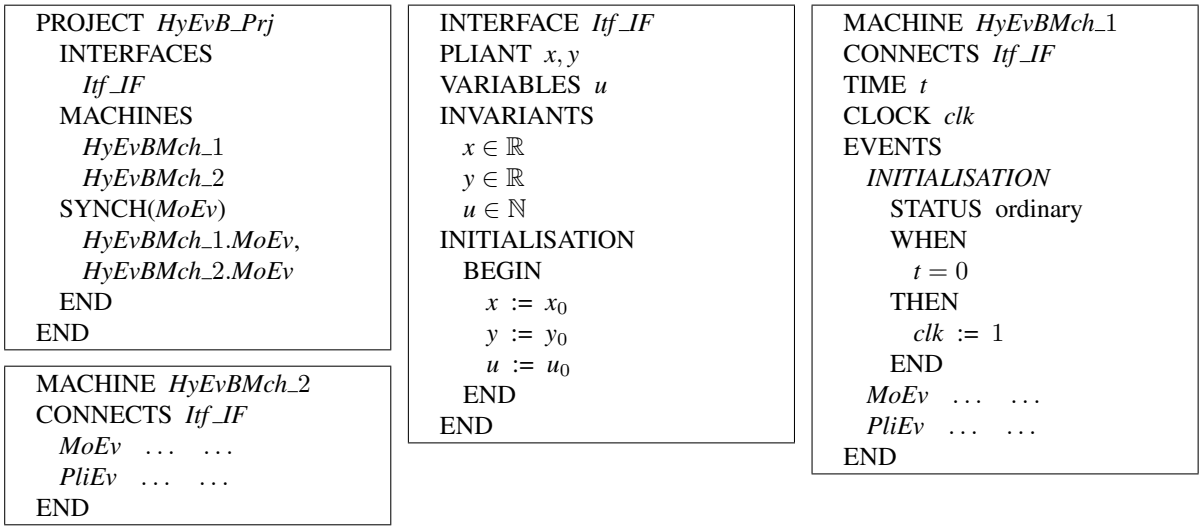

Figure 3: A schematic Hybrid Event-B project. The PROJECT file names the constituent machines and interfaces, and defines the project-wide mode event SYNCHronisations. The INTERFACE file declares shared variables, their intialisations, and any invariants that involve them.

4. Multiple Hybrid Event-B Machines

The principal objective in modelling complex systems in the B-Method is to start with small simple descriptions and to refine to richer, more detailed ones. This means that, at the highest levels of abstraction, the modelling must **abstract away from concurrency**. By contrast, at lower levels of abstraction, the events describing detailed individual behaviours of components become visible. Thus an integrated representation risks hitting the combinatorial explosion of needing to represent each possible combination of concurrent activities within a separate event, and there is a much stronger incentive to put each (relatively) independent component into its own machine, synchronised appropriately. To put it another way, there is a very strong incentive to **not abstract away from concurrency**, an impulse that matches with the actual system architecture. Thus to model large systems, multi-machine configurations are certainly desirable. At minimum, they partition the functionality, allowing limited focus and independent working. We introduce the essentials of multi-machine working, referring to [19] for a fuller description.

Fig. 3 shows a schematic multi-machine project, building on Fig. 2. The PROJECT file names the syntactic constituents of the project: interfaces and machines (also synchronisations, described later). The INTERFACE file declares shared variables, their intialisations, and any invariants that contain an occurrence of any of their variables.[3] In Fig. 3, the variables of Fig. 2, their intialisations, and the invariants have been made shared, and have been moved into interface *Itf_IF*. This leaves the machines *HyEvBMch_1* and *HyEvBMch_2* with just time and clock declarations, and their events.

The structural rules that govern projects largely follow from the scope rules for the various identifiers present. These are as follows. The scopes of INTERFACE names, MACHINE names and variable names are project-wide. The scopes of event names are their containing machine (thus *HyEvBMch_1.MoEv* uniquely identifies the *MoEv* belonging to *HyEvBMch_1*), and the scopes of inputs, locals and outputs are their containing event (so *HyEvBMch_1.MoEv.l* uniquely identifies local *l* of *HyEvBMch_1.MoEv*).

Since the verification process demands that events preserve invariants, a MACHINE must have a CONNECTS *itf* clause for every interface *itf* that has a variable that any of its events need; READS *itf* provides read-only access if that is sufficient. Evidently the connected interfaces also contain all the

[3] The INTERFACE concept is adapted from the similarly named idea in [31].

invariants that any event of the machine must preserve. A more detailed description of the structural constraints governing multi-machine projects (the diamond rules) appears in [19], but they amount to little more than an elaboration of what just been stated.

In a multi-machine project, *all* the machines run concurrently, all the time. Pliant events execute in each machine separately. Mode events also execute in each machine separately, unless they occur in a SYNCH clause in the PROJECT file. The SYNCHronisation clause specifies that all the mode events named in it must execute simultaneously, effectively aggregating them into a single project-wide mode event.[4] Evidently, simultaneously executing events of any kind must not specify conflicting updates to variables, so we assume that the updates in the two *PliEv*'s of *HyEvBMch_1* and *HyEvBMch_2* of Fig. 3 are disjoint (because they execute concurrently), having, for example, arisen by partitioning the updates in the *PliEv* of Fig. 2. Likewise, we can imagine that the provisions of the *BAPred* of Fig. 2 have also been split into the updates in the two *MoEv*'s of *HyEvBMch_1* and *HyEvBMch_2* of Fig. 3 (since they are required to execute simultaneously because of the SYNCH clause in the project file).

4.1. Semantics of Multiple Hybrid Event-B Machines

The semantics of multiple Hybrid Event-B machines is built, as much as possible around the semantics of single machines. Thus the key concept is again the system trace, recording the values of the variables of all machines throughout a run. Additionally, a system trace contains a record of which pliant event is being executed in each machine at all times during the run.

Instead of SYNCHronisations (which are syntactic), the definition in [19] uses **clusters** of mode events, which are simply arbitrary sets of mode events containing at most one from any machine. The definition does not specify how clusters arise. (This enables different syntactic synchronisation mechanisms to be defined on top of the semantics, if desired.) A cluster is enabled iff all its events are enabled.

The construction of system traces proceeds as in the outline given in Section 3.2, with minor modifications. Thus, after initialisation, enabled pliant events are selected in all machines and start to execute. The execution continues until a mode event (cluster) is enabled, at which point pliant event execution is paused and the mode event cluster is executed. Upon its completion, it is known which machines specified the events involved in the cluster just executed — those machines select a fresh pliant event to execute. For the remaining machines, the pliant event execution pause is a technical artifice, and they resume execution of their preceding pliant event, whose identity is recorded in the system trace, as noted above. Execution continues in this manner, with mode event cluster execution alternating with plaint event execution/continuation. Details are in [19].

That the procedure outlined does not fail during the construction of multi-machine runs can be guaranteed by confirming the multi-machine Hybrid Event-B POs. And, just as multi-machine semantics is a minimal departure from single machine semantics, so the multi-machine POs are the same as the single machine POs, except that, where a cluster forces the consideration of events from more than one machine, all the events of all the machines in question must, in principle, be taken into account simultaneously. (The fact that pliant events cannot be clustered helps enormously.) There is a careful discussion in [19].

The single machine POs, suitably reinterpreted as indicated, together with a small number of additional conditions, enable the correctness of multi-machine Hybrid Event-B projects to be proved (Theorem 12.4 of [19]).

4.2. Multiple Hybrid Event-B Machine Refinement

Each machine in a multi-machine project executes independently unless coupled to other machines via shared variables and synchronised events. This default independence forces a fairly restrictive disci-

[4] To prevent specifying complicated mutual exclusion protocols by stealth, SYNCH clauses in a project must be disjoint.

pline on the refinement of machines and interfaces. Thus, each separate abstract machine must be refined to a corresponding separate concrete machine, with a retrieve relation restricted to the variables of the two machines. And each separate abstract interface must be refined to a corresponding separate concrete interface, with a retrieve relation restricted to the variables of the two interfaces. Thus, the project-wide retrieve relation is some subrelation of a cartesian product of universal relations, one for each refining pair of components.

5. Decomposition and Verification

Above we discussed multi-machine projects assuming them given *a priori*. However, taking an existing machine and decomposing it into a multi-machine project (or decomposing a machine within a larger project) requires additional considerations.

Partitioning for Hybrid Event-B is discussed in [19], where the partitioning process is designed as a purely syntactic transformation of a large machine into a collection of smaller machines. This simple picture was intended to not divert attention from the main job of [19], which was the proof that the single machine POs, suitably reinterpreted and with little additional support, guaranteed correctness of any execution.

Still, it is not hard to see that purely syntactic manipulations can grow increasingly complicated, and eventually can get sufficiently involved that they become more akin to development steps requiring proof, than to pure syntax rearrangement. In this section, we take such an approach to the partitioning process, making it more flexible, at the price of requiring the verification of new POs to establish its soundness in the context of any specific system model.

We start with the conditions constraining the decomposition process as discussed in [19]. These state that if a machine M is part of a project $\mathcal{P}$, and M is decomposed into submachines $M_1 \ldots M_k$ with new interfaces $Itf_1 \ldots Itf_l$, then firstly: the new components must adhere to the general structural conditions governing collections of machines and interfaces in a project as overviewed in Section 4; and secondly, the following more specific conditions must also hold.

[♦20] Every pliant event *PliEv* of M is decomposed into subevents $PliEv_1 \ldots PliEv_k$, one for each submachine $M_1 \ldots M_k$, with each subevent having the same INIT and WHERE guards as *PliEv*, and with the assigning clauses appropriately distributed among the $PliEv_1 \ldots PliEv_k$.

[♦21] For every pair of distinct pliant events of M, the (pairwise) conjunction of its WHERE guards is unsatisfiable.

[♦22] For every pair $MoEv1, MoEv2$ of distinct mode events of M, if the (pairwise) conjunction of its WHERE guards is satisfiable, then there is a submachine M_j of the decomposition of M, such that some subevent of the synchronised decomposition of $MoEv1$ and some subevent of the synchronised decomposition of $MoEv2$ (or, in each case, the event itself if the event is undecomposed), are both declared in M_j.

It is easiest to discuss these in reverse order. Regarding mode events, if a mode event *MoEv* is syntactically decomposed into two subevents *MoEvX* and *MoEvY*, then if *MoEvX* and *MoEvY* are synchronised within the relevant project file, it is evidently much easier to ensure that their collective effect is the same as that of *MoEv*. So we always assume that that is the case.

As in [19], we need to ensure that the result of decomposition has the same properties as the original machine regarding: enabledness of (synchronised) events, updates to the state, and scheduling choices of (synchronised) events.

Regarding mode events' enabledness, the synchronisation assumption means that it is sufficient to distribute the clauses of *MoEv*'s guard between the guards of *MoEvX* and *MoEvY* to ensure that their conjunction is equivalent to the guard of *MoEv*. However, we can weaken this requirement to the PO:

$$Inv(u) \Rightarrow [\ \forall i? \bullet grd_{MoEv}(u, i?) \Leftrightarrow grd_{MoEvX}(u_X, i?_X) \wedge grd_{MoEvY}(u_Y, i?_Y)\ ] \tag{1}$$

In (1), u refers to the part of the overall system state relevant to *MoEv*, and u_X, u_Y refer to those parts of u needed by *MoEvX* and *MoEvY* respectively — likewise for the inputs $i?$ of *MoEv* and *MoEvX*, *MoEvY*. *Inv* refers to all the invariants that concern the state variables that appear in *MoEv*, *MoEvX*, *MoEvY*. The invariants will be distributed in various ways among the machines and interfaces of $\mathcal{P}$ and its decomposed version, but can always be identified unambiguously since invariants are never transformed during decomposition; they may merely be moved from place to place, according to the structural rules (which demand that any invariant resides in the interface or machine containing all the variables that occur in it).

The PO (1) is strictly more powerful than mere distribution of the clauses of *MoEv*'s guard because the equivalence is only required to hold when *Inv* is true. We assume that the undecomposed machine M has been proved correct, so that *Inv* truly denotes an invariant set of the system, and is thus a safe over-approximation of the reachable subset of the state space. This observation may permit some non-trivial transformation of *MoEv*'s guard when *MoEvX* and *MoEvY* are constructed. The biimplication in (1) ensures equivalence between the enabledness of the undecomposed and decomposed systems. And equivalence of enabledness in any given state implies the same scheduling choices in that state, provided **[♦22]** holds to enforce appropriate mutual exclusion between different synchronisations.

The argument just detailed applies analogously to state update. Accordingly, we get the PO:

$$\begin{aligned} Inv(u) \Rightarrow [\ \forall u', o! \bullet\ & BApred_{MoEv}(u, i?, u', o!) \Leftrightarrow \\ & BApred_{MoEvX}(u_X, i?_X, u'_X, o!_X) \wedge BApred_{MoEvY}(u_Y, i?_Y, u'_Y, o!_Y)\ ] \end{aligned} \tag{2}$$

In the overwhelming majority of cases, the *BApred* of a mode event is expressed using a collection of individual state updates $x := E$. In such cases (2) offers no additional generality since all that can be done is to move such assignments into the relevant subevent. Nevertheless, the more general possibility exists. With the same state updates guaranteed, and the same enabledness and scheduling possibilities in all states, the equivalence of the undecomposed and decomposed systems follows relatively trivially when only mode events are decomposed.

Pliant events obey different rules. For predominantly physical reasons pliant events are unsynchronised, a design decision that has an impact. Specifically, absence of synchronisation implies that pieces of different decomposed (but simultaneously enabled) events could not be prevented from being executed simultaneously by a nondeterministic scheduling policy, a possibility that would destroy semantic equivalence. Hence we introduced the restriction in **[♦21]**. The same facts also imply that each piece of a decomposed pliant event must be enabled exactly when the undecomposed event is; hence we have **[♦20]**. Following our line of reasoning above, we can relax **[♦20]** to:

$$Inv(u) \Rightarrow [\ grd_{PliEv}(u) \wedge iv_{PliEv}(u) \Leftrightarrow grd_{PliEvX}(u_X) \wedge iv_{PliEvX}(u_X)\ ] \tag{3}$$

$$Inv(u) \Rightarrow [\ grd_{PliEv}(u) \wedge iv_{PliEv}(u) \Leftrightarrow grd_{PliEvY}(u_Y) \wedge iv_{PliEvY}(u_Y)\ ] \tag{4}$$

In (3) and 4) we assume we are dealing with a pliant event *PliEv* which is to be decomposed into *PliEvX* and *PliEvY*. The fact that there are two statements is another reflection of the absence of synchronisation.

In **[♦20]** it states that 'the assigning clauses [are] appropriately distributed among' $PliEv_X$ and $PliEv_Y$ (for our case). When the assigning clauses are ODEs and direct assignments, this definition is as uncontroversial as in the mode event case. However, its implications are potentially more subtle for COMPLY clauses.

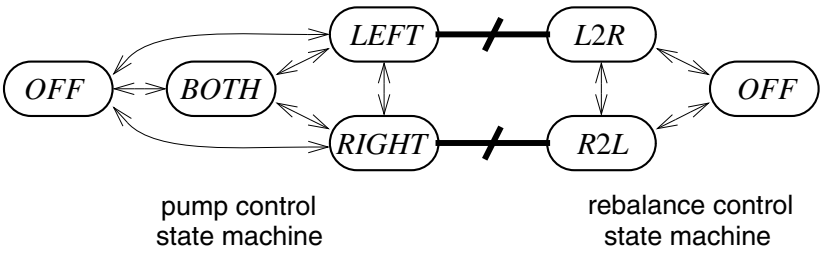

Figure 4: The top level fuel delivery system transition diagram. The pump state diagram is on the left and the rebalance control state diagram is on the right. The heavy crossed lines connect the only forbidden pairs of states. Otherwise, every pair of states, and every transition involving one or other of the pump or rebalance controls, is permitted.

Suppose $Cmply_{PliEv}(u, i?)$ is the comply clause of *PliEv*. We assume that $Cmply_{PliEv}(u, i?)$ can be rearranged into an exclusive-or normal form (by reducing to disjunctive normal form first, and then manipulating further, if necessary): $Cmply_{PliEv}(u, i?) \equiv \bigoplus_{j \in \mathsf{J}} Cmply_{PliEv,j}(u, i?)$. This kind of structure represents a typical modelling case analysis in logical form. Considering $PliEv_X$ and $PliEv_Y$, we analogously assume that $Cmply_{PliEvX}(u_X, i?_X) \equiv \bigoplus_{j_X \in \mathsf{J}_X} Cmply_{PliEvX,j_X}(u_X, i?_X)$ and that $Cmply_{PliEvY}(u_Y, i?_Y) \equiv \bigoplus_{j_Y \in \mathsf{J}_Y} Cmply_{PliEvY,j_Y}(u_Y, i?_Y)$.

We express our liberalisation of **[♦20]** by assuming that for each term $Cmply_{PliEv,j}$ of $Cmply_{PliEv}$ there are unique terms $Cmply_{PliEvX,j_X}$ in $Cmply_{PliEvX}$ and $Cmply_{PliEvY,j_Y}$ in $Cmply_{PliEvY}$ such that $Cmply_{PliEv,j}$ is equivalent to the conjunction of $Cmply_{PliEvX,j_X}$ and $Cmply_{PliEvY,j_Y}$. Calling the function on the index sets implied by the preceding statement Z, we have:

$$Z : \mathsf{J} \to \mathsf{J}_X \times \mathsf{J}_Y \tag{5}$$

$$Inv(u) \Rightarrow [\, \forall j \in \mathsf{J} \bullet \forall\, i? \bullet Cmply_{PliEv,j}(u, i?) \Leftrightarrow Cmply_{PliEvX,\pi_X(Z(j))}(u_X, i?_X) \wedge Cmply_{PliEvY,\pi_Y(Z(j))}(u_Y, i?_Y) \,] \tag{6}$$

In (6), the universal quantifier $\forall j \in \mathsf{J}$ is written bold to underline that it quantifies over syntactic items rather than semantic values; π_X and π_Y are the projectors to the first and second components of $Z(j)$. The fact that for each $u, i?$, only one $Cmply_{PliEv,j}$ term can be true follows from the assumed exclusive-or normal form. Of course, (5) and (6) give just one way of ensuring that $Cmply_{PliEv}(u, i?)$ and $Cmply_{PliEvX}(u_X, i?_X) \wedge Cmply_{PliEvY}(u_Y, i?_Y)$ are equivalent, which would be sufficient to ensure the equivalence we need. Other schemes that achieve this would also be acceptable. As the case study below develops, we will see the utility of some of these more flexible rules in its various stages.

6. Top Level Fuel System Models

We now embark on the modelling of the fuel system in Hybrid Event-B, and on uncovering the insights this can offer. The state machine view of the fuel supply system is shown in Fig. 4. This consists of two state machines, corresponding to the pump control and the rebalance control. The overall state machine is the product of these two, aside from the two forbidden states indicated by the heavy struck through lines. For the sake of simplicity, we will assume that these state machines are implemented in the cockpit by a set of four press-and-latch buttons for the pump control, and a set of three press-and-latch buttons for the rebalance control, with, in addition, a mechanical interlock to prevent the engagement of the forbidden states. We assume that pressing-and-latching any button of either set causes the release of the previously depressed button from the set.

The state level view merely reflects the changes of configuration of the fuel system that can be effected by the pilot. And although we have described it in purely mechanical terms, there is, of course, no reason that such state control should not be implemented digitally in a modern light aircraft.

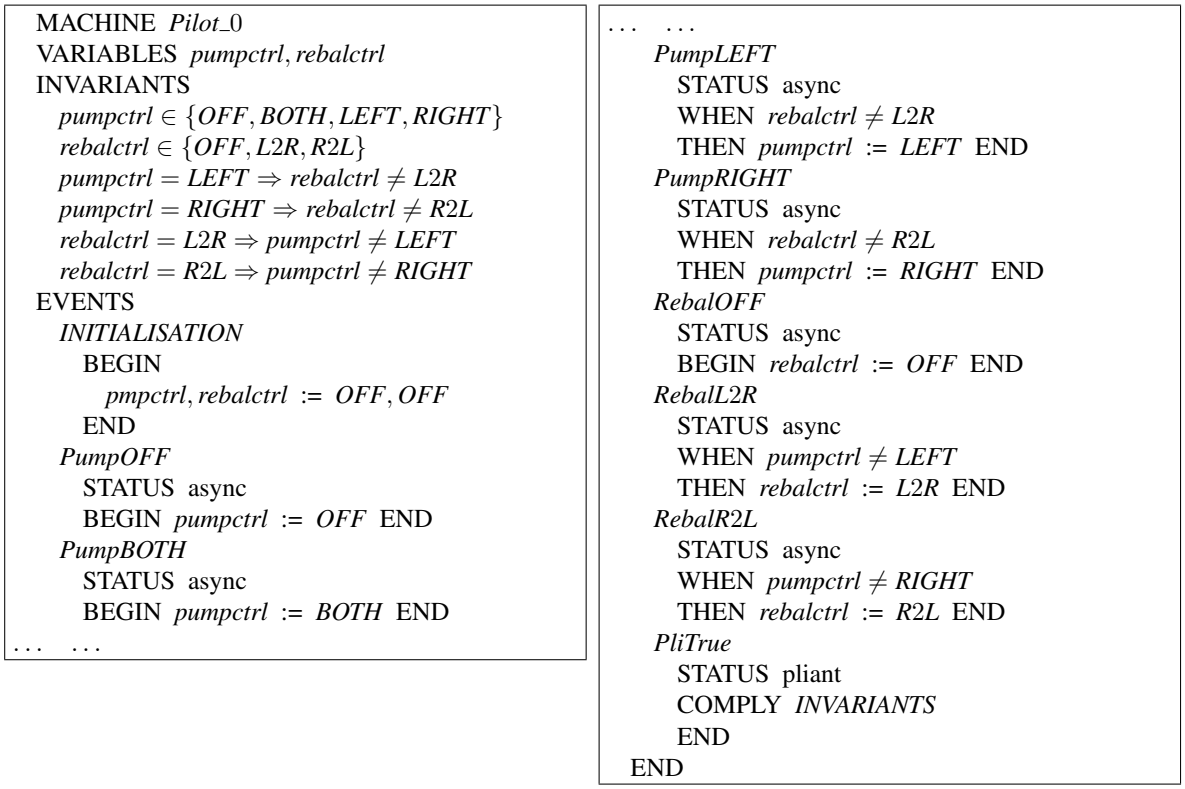

Figure 5: The top level Hybrid Event-B model of the fuel delivery system. The pilot's view.

The mode level view, being essentially the view seen by the pilot, is captured in the Hybrid Event-B machine *Pilot_0* of Fig. 5. The machine has two variables, *pumpctrl* and *rebalctrl*, with the obvious meanings, and the values each can take are described in the first two lines of the INVARIANTS clause. The remaining invariants describe the forbidden configurations. The remainder of the machine describes the EVENTS that are available. There are events to manipulate the fuel control: *PumpOFF*, *PumpBOTH*, *PumpLEFT*, *PumpRIGHT*; and events to manipulate the rebalance control: *RebalOFF*, *RebalL2R*, *RebalR2L*. These are all *mode events* in Hybrid Event-B parlance, i.e. they specify instantaneous changes of state at isolated moments of time. To this extent they are just like (conventional) Event-B events, and the notation is deliberately kept the same.

Events' STATUS declarations distinguish mode events from pliant events, and record other pragmatic properties of events. The most important of these is the 'async' property, which allows mode events to execute lazily (i.e. *not* as soon as they are enabled). This is a shorthand that removes the need for an input, which is the normal trick that specifies lazy execution (see **[5]** in Section 3.2, and [18]). In the case of the present application, these async events would be stimulated from the environment (i.e. by the pilot) and not by the immediate action of some physical law.

Furthermore, since Hybrid Event-B describes the behaviour at *all* moments of time, each Hybrid Event-B machine must have at least one *pliant event*, to cover continuous behaviour between the isolated mode events. In *Pilot_0* this duty is covered by the *PliTrue* event, which simply stipulates default compliance with the *INVARIANTS* any time a mode event is not executing.

A certain amount of previous experience [17, 16, 13, 15], has shown that focusing first on a mode description of a desired design is highly beneficial in organising the refinement based development of a complex hybrid system in a perspicuous manner. In the present case we follow the same strategy, but notice first that the mode level description we gave is not yet at the pumps and valves level of the

PROJECT *FuelPump_Prj_1*
[REFINES *FuelPump_Prj_0*]
INTERFACES
Central_IF_1
MACHINES
Pilot_1
Controller_1
SYNCH(*PumpOFF*)
Pilot_1.PumpOFF_S,
Controller_1.PumpOFF_S
END
SYNCH(*PumpBOTH*)
Pilot_1.PumpBOTH_S,
Controller_1.PumpBOTH_S
END
SYNCH(*PumpLEFT*)
Pilot_1.PumpLEFT_S,
Controller_1.PumpLEFT_S
END
SYNCH(*PumpRIGHT*)
Pilot_1.PumpRIGHT_S,
Controller_1.PumpRIGHT_S
END
SYNCH(*RebalOFF*)
Pilot_1.RebalOFF_S,
Controller_1.RebalOFF_S
END
SYNCH(*RebalL2R*)
Pilot_1.RebalL2R_S,
Controller_1.RebalL2R_S
END
SYNCH(*RebalR2L*)
Pilot_1.RebalR2L_S,
Controller_1.RebalR2L_S
END
END

INTERFACE *Central_IF_1*
VARIABLES
$pumpctrl, rebalctrl,$
$pumpP_L, pumpP_R, valveL, valveR,$
$valveV_{L1}, valveV_{L2}, valveV_{R1}, valveV_{R2}$
INVARIANTS
$pumpctrl \in \{OFF, BOTH, LEFT, RIGHT\}$
$rebalctrl \in \{OFF, L2R, R2L\}$
$pumpctrl = LEFT \Rightarrow rebalctrl \neq L2R$
$pumpctrl = RIGHT \Rightarrow rebalctrl \neq R2L$
$rebalctrl = L2R \Rightarrow pumpctrl \neq LEFT$
$rebalctrl = R2L \Rightarrow pumpctrl \neq RIGHT$
$pumpP_L, pumpP_R \in \{OFF, ON\}$
$valveL, valveR,$
$valveV_{L1}, valveV_{L2}, valveV_{R1}, valveV_{R2}$
$\in \{CL, OP\}$
$pumpctrl = OFF \Leftrightarrow$
$pumpP_L = OFF \wedge pumpP_R = OFF \wedge$
$valveL = CL \wedge valveR = CL$
$pumpctrl = LEFT \Leftrightarrow$
$pumpP_L = ON \wedge pumpP_R = OFF \wedge$
$valveL = OP \wedge valveR = CL$
$pumpctrl = RIGHT \Leftrightarrow$
$pumpP_L = OFF \wedge pumpP_R = ON \wedge$
$valveL = CL \wedge valveR = OP$
$pumpctrl = BOTH \Leftrightarrow$
$pumpP_L = ON \wedge pumpP_R = ON \wedge$
$valveL = OP \wedge valveR = OP$
$rebalctrl = OFF \Leftrightarrow$
$valveV_{L1} = CL \wedge valveV_{R1} = CL \wedge$
$valveV_{L2} = CL \wedge valveV_{R2} = CL$
$rebalctrl = L2R \Leftrightarrow$
$valveV_{L1} = CL \wedge valveV_{R1} = OP \wedge$
$valveV_{L2} = OP \wedge valveV_{R2} = CL$
$rebalctrl = R2L \Leftrightarrow$
$valveV_{L1} = OP \wedge valveV_{R1} = CL \wedge$
$valveV_{L2} = CL \wedge valveV_{R2} = OP$
INITIALISATION
BEGIN
$pmpctrl, rebalctrl := OFF, OFF$
$pumpP_L, pumpP_R, valveL, valveR :=$
OFF, OFF, CL, CL
$valveV_{L1}, valveV_{L2}, valveV_{R1}, valveV_{R2}, :=$
CL, CL, CL, CL
END
END

Figure 6: Level 1 Hybrid Event-B project for the fuel delivery system. Introduction of the valves and pumps. Here is the *FuelPump_Prj_1* PROJECT file and the INTERFACE *Central_IF_1*.

MACHINE *Pilot_1*
REFINES *Pilot_0*
CONNECTS *Central_IF_1*
EVENTS
PumpOFF_S
REFINES *PumpOFF*
STATUS async
BEGIN *pumpctrl* := *OFF* END
PumpBOTH_S
REFINES *PumpBOTH*
STATUS async
BEGIN *pumpctrl* := *BOTH* END
PumpLEFT_S
REFINES *PumpLEFT*
STATUS async
WHEN $rebalctrl \neq L2R$
THEN *pumpctrl* := *LEFT* END
PumpRIGHT_S
REFINES *PumpRIGHT*
STATUS async
WHEN $rebalctrl \neq R2L$
THEN *pumpctrl* := *RIGHT* END
RebalOFF_S
REFINES *RebalOFF*
STATUS async
BEGIN *rebalctrl* := *OFF* END
RebalL2R_S
REFINES *RebalL2R*
STATUS async
WHEN $pumpctrl \neq LEFT$
THEN *rebalctrl* := *L2R* END
RebalR2L_S
REFINES *RebalR2L*
STATUS async
WHEN $pumpctrl \neq RIGHT$
THEN *rebalctrl* := *R2L* END
PliTrue
STATUS pliant
COMPLY *INVARIANTS*
END
END

MACHINE *Controller_1*
CONNECTS *Central_IF_1*
EVENTS
PumpOFF_S
BEGIN
$pumpP_L, pumpP_R, valveL, valveR$:=
OFF, OFF, CL, CL
END
PumpBOTH_S
BEGIN
$pumpP_L, pumpP_R, valveL, valveR$:=
ON, ON, OP, OP
END
PumpLEFT_S
WHEN $\neg(valveV_{L1} = CL \wedge valveV_{R1} = OP)$
THEN
$pumpP_L, pumpP_R, valveL, valveR$:=
ON, OFF, OP, CL
END
PumpRIGHT_S
WHEN $\neg(valveV_{L1} = OP \wedge valveV_{R1} = CL)$
THEN
$pumpP_L, pumpP_R, valveL, valveR$:=
OFF, ON, CL, OP
END
RebalOFF_S
BEGIN
$valveV_{L1}, valveV_{R1}, valveV_{L2}, valveV_{R2}$:=
CL, CL, CL, CL
END
RebalL2R_S
WHEN $\neg(pumpP_L = ON \wedge pumpP_R = OFF)$
THEN
$valveV_{L1}, valveV_{R1}, valveV_{L2}, valveV_{R2}$:=
CL, OP, OP, CL
END
RebalR2L_S
WHEN $\neg(pumpP_L = OFF \wedge pumpP_R = ON)$
THEN
$valveV_{L1}, valveV_{R1}, valveV_{L2}, valveV_{R2}$:=
OP, CL, CL, OP
END
PliTrue
STATUS pliant
COMPLY *INVARIANTS*
END
END

Figure 7: Level 1 Hybrid Event-B project for the fuel delivery system. Introduction of the valves and pumps. The two machines, *Pilot_1* and *Controller_1*.

description in Section 2, so is not yet good to interface with the more physical behaviour we wish to capture in this case study.

In this paper, we develop the fuel supply application by combining component machines for all the different spheres of activity. Accordingly, we next introduce the *Controller_1* machine which manipulates the pumps and valves at the pilot's behest.

Given that we have more than one machine, we need the full apparatus of the multi-machine PROJECT formalism, omitted hitherto. Fig. 6 shows the *FuelPump_Prj_1* PROJECT file and the *Central_IF_1* INTERFACE. The project shows a hypothetical refinement of an earlier project *FuelPump_Prj_0*, which would exist had we indeed packaged the *Pilot_0* machine inside a project. As well as now containing the previously introduced variables, the interface contains the new variables for the valves and pumps. It also contains the invariants that connect these to the pump control variables.

Fig. 7 shows the two machines in the project, *Pilot_1* and *Controller_1*. The former refines *Pilot_0*. Notice how the presence of the interface construct has removed a large part of the content of *Pilot_0*. REFINES clauses in the events of *Pilot_1* state which parent events in *Pilot_0* they refine. *Controller_1* is new, and describes how the events manipulate the valve and pump variables.

The two machines *Pilot_1* and *Controller_1* must act in harmony. This is enforced by the SYNCH clauses of the *FuelPump_Prj_1* project file. For example, synchronisation SYNCH(*PumpOFF*) requires that event *PumpOFF_S* in machine *Pilot_1* executes simultaneously with event *PumpOFF_S* in machine *Controller_1*. And so on. The '*_S*' suffixes on the event names are for readability, to indicate participation in a synchronisation, but have no semantic significance.

The newly introduced variables are named by analogy with the description in Section 2. Pumps are either *OFF* or *ON*, while valves are either *CL*osed or *OP*en. After declaring the variables and their allowed values, the remainder of the new INVARIANTS in *Central_IF_1* are *joint invariants*, concerned with expressing the relationship between the *Pilot_0/1* variables and the *Controller_1* variables (in traditional Event-B manner).

Our assumptions about how the controls work result in a relatively simple correspondence between pilot controls and settings of the pumps and valves. The joint invariants make clear that the fuel control can be implemented using just the pumps and their valves to the collector (L for pump P_L), while the rebalancing control can be implemented using the various $V_$ valves. This makes for a particularly clean design. It is easy to imagine that if the pipework depicted in Fig. 1 were controlled in a different way, then the correspondence between the two levels could come out more complicated.

It has to be admitted that the clean design is partly a consequence of deliberate oversimplification. Thus the only practical way of achieving fuel rebalancing is if the pumps are *BOTH* on. That way part (but only part) of the flow of one pump is diverted to refilling the other tank. But we might wish to rebalance on the ground, without the other pump running. Or we might wish (in the air) to feed the engine using one pump and use the other pump exclusively for rebalancing. Both scenarios are impossible in our setup since they couple the state of the *L* and *R* valves to the state of *rebalctrl*. Representing such things would be entirely possible, at the cost of a more detailed, longer description. We avoid doing so for simplicity and brevity.

7. Physical Behaviour of the Pumps

In Fig. 8 we find the next level of detail in our development. At the top is the new project file *FuelPump_Prj_2*. It introduces the *FuelFlow_2* machine, but is otherwise unchanged from *FuelPump_Prj_1* which it refines.

The *FuelFlow_2* machine READS the existing interface. This denotes that it only needs read access to the variables declared in *Central_IF_1*, and does not introduce any new invariants involving those

variables. (From the mechanical verification point of view, this makes the access by *FuelFlow_2* to the interface lighter.)

The *FuelFlow_2* machine introduces some pliant variables to represent the continuous behaviour. We focus exclusively on the flow rates in the pipework of the model of Fig. 1. The fuel flow rate delivered to the engine is flrd_E. The raw pumping rates of the left and right pumps are flr_L and flr_R respectively. The fuel rates delivered to the engine by the left and right pumps are flrd_L and flrd_R respectively, leading the equation $\mathit{flrd}_E = \mathit{flrd}_L + \mathit{flrd}_R$, reflecting the fact that the fuel is assumed to be an *incompressible* fluid. This equation could be an invariant, were it not for the fact that the same equation appears in the COMPLY clause of the *SupplyEngine* pliant event, making its presence as an invariant superfluous. The excess of the raw fuel rates over the demanded rates, which are fed back to the left and right tanks are expressed by flrfb_L and flrfb_R for the two pumps respectively. The rebalancing flow rates are flr_{L2R} and flr_{R2L}. So $\mathit{flr}_L = \mathit{flrd}_L + \mathit{flrfb}_L$ or $\mathit{flr}_L = \mathit{flrd}_L + \mathit{flr}_{L2R} + \mathit{flrfb}_L$ is true for the left pump, depending on whether flr_{L2R} is nonzero (and assuming we only mention flr_{L2R} when it is nonzero). Similarly for the right pump.

The invariants of *FuelFlow_2* give some of the properties of these flow rates. Thus they are all real numbers, and are either zero (if the relevant part of the apparatus is switched off), or within a real interval. For the engine, the interval lies between ERT_{MIN} and ERT_{MAX}. For the raw pumping rates, the interval lies between RPRT_{MIN} and RPRT_{MAX}, which is assumed to be a fairly narrow real interval reflecting the relatively fixed rate at which the pumps work. The various dependent flow rates, when nonzero, are assumed to lie in a broader interval $[\mathit{PRT}_{MIN} \ldots \mathit{PRT}_{MAX}]$, reflecting the variability of engine demand, and of the different possibilities for directing the fuel round the system. All these variables are initialised to 0. Note that we do not write e.g. $\mathit{flr}_E(t)$ — the time dependence is an automatic consequence of the PLIANT declaration. (N. B. Mode variables are also functions of time, albeit piecewise constant ones.)

We turn to the pliant event *SupplyEngine*. Recalling that the fuel is an incompressible fluid, fuel entering the pipework is instantaneously balanced by fuel leaving the pipework. Thus, the semantics of the pipework system is overwhelmingly one of equality between various quantities. However, the relative dependencies between the various quantities are less clear. The engine demands as much fuel as it needs to function at the power the pilot requests. The pumps, when switched on, wish to pump as hard as their mechanical specification stipulates. Aside from pilot demand, the extent to which they are able to push fuel through the pipework depends also on which flows though the pipework are available.

The ANY clause of *FlyAircraft* introduces a number of quantities. All are implicitly time dependent. flr_E^{CH} is the chosen fuel rate corresponding to the pilot's request; it is constrained to the same values as flr_E in the WHERE clause. The other quantities, in pairs, describe how pairs of flows which meet at a single place must be constrained. Thus they are all values in the open interval $(0 \ldots 1)$ (hence all are nonzero), and pairwise, they sum to 1 (reflecting incompressibility), with an additional constraint concerning their relative magnitude. Thus c_L and c_R describe how the raw outputs of the left and right pumps are scaled back when both are feeding the engine (with the remaining pump outputs flrfb_L and flrfb_R fed back to the relevant tanks). They sum to 1, and do not differ by much $|\, c_L - c_R \,| < H$, reflecting our expectation that the two pumps are physically similar. The quantities $c_{L2R,E}$ and $c_{L2R,R}$ describe how the output of the left pump is divided between feeding the engine ($c_{L2R,E}$) and filling the right tank ($c_{L2R,R}$), when rebalancing is set to *L2R* and the left pump is working. Here we expect the rebalancing to outweigh feeding the engine, reflected in the constraint $c_{L2R,E}/c_{L2R,R} < H$. Similarly for $c_{R2L,E}$ and $c_{R2L,L}$. The same constant H is used for all these constraints, for simplicity.

At the present level of modelling, the dynamics of the fuel system is still very nondeterministic. The COMPLY clause stipulates what is defined. The first line stipulates that the fuel rate delivered to the engine, flrd_E, must be the rate chosen by the pilot, flr_E^{CH}, according to how the aircraft is being flown. The next line says that flrd_E is the sum of the delivered fuel rates from the two pumps, $\mathit{flrd}_L + \mathit{flrd}_R$.

The lines after that treat the case when both pumps are switched off. Then, there is no raw output from

PROJECT *FuelPump_Prj_2*
REFINES *FuelPump_Prj_1*
INTERFACES
Central_IF_1
MACHINES
... ...

... ...
Pilot_1
Controller_1
FuelFlow_2
SYNCH(*PumpOFF*) ... ...
END

MACHINE *FuelFlow_2*
READS *Central_IF_1*
PLIANT $flrd_E, flrd_L, flrd_R, flr_L, flr_R, flrfb_L, flrfb_R, flr_{L2R}, flr_{R2L}$
INVARIANTS
$flrd_E \in \mathbb{R} \wedge flrd_E \in \{0\} \cup [ERT_{MIN} \ldots ERT_{MAX}]$
$flrd_L \in \mathbb{R} \wedge flrd_L \in \{0\} \cup [PRT_{MIN} \ldots PRT_{MAX}]$
$flrd_R \in \mathbb{R} \wedge flrd_R \in \{0\} \cup [PRT_{MIN} \ldots PRT_{MAX}]$
$flr_L \in \mathbb{R} \wedge flr_L \in \{0\} \cup [RPRT_{MIN} \ldots RPRT_{MAX}]$
$flr_R \in \mathbb{R} \wedge flr_R \in \{0\} \cup [RPRT_{MIN} \ldots RPRT_{MAX}]$
$flrfb_L \in \mathbb{R} \wedge flrfb_L \in \{0\} \cup [PRT_{MIN} \ldots PRT_{MAX}]$
$flrfb_R \in \mathbb{R} \wedge flrfb_R \in \{0\} \cup [PRT_{MIN} \ldots PRT_{MAX}]$
$flr_{L2R} \in \mathbb{R} \wedge flr_{L2R} \in \{0\} \cup [PRT_{MIN} \ldots PRT_{MAX}]$
$flr_{R2L} \in \mathbb{R} \wedge flr_{R2L} \in \{0\} \cup [PRT_{MIN} \ldots PRT_{MAX}]$
EVENTS
INITIALISATION
BEGIN
$flrd_E, flrd_L, flrd_R, flr_L, flr_R, flrfb_L, flrfb_R, flr_{L2R}, flr_{R2L} := 0,0,0,0,0,0,0,0,0$
END
SupplyEngine
STATUS pliant
ANY $flr_E^{CH}, c_L, c_R, c_{L2R,E}, c_{L2R,R}, c_{R2L,E}, c_{R2L,L}$
WHERE
$flr_E^{CH} \in \mathbb{R} \wedge$
$flr_E^{CH} \in \{0\} \cup [ERT_{MIN} \ldots ERT_{MAX}] \wedge$
$\{c_L, c_R, c_{L2R,E}, c_{L2R,R}, c_{R2L,E}, c_{R2L,L}\} \subset \mathbb{R} \wedge$
$\{c_L, c_R, c_{L2R,E}, c_{L2R,R}, c_{R2L,E}, c_{R2L,L}\} \subset (0 \ldots 1) \wedge$
$c_L + c_R = 1 \wedge |c_L - c_R| < H \wedge$
$c_{L2R,E} + c_{L2R,R} = 1 \wedge c_{L2R,E}/c_{L2R,R} < H \wedge$
$c_{R2L,E} + c_{R2L,L} = 1 \wedge c_{R2L,E}/c_{R2L,L} < H$
... ...

... ...
COMPLY
$flrd_E = flr_E^{CH} \wedge$
$flrd_E = flrd_L + flrd_R \wedge$
$[pumpctrl = OFF \Rightarrow$
$flrd_L = flrfb_L = flr_L = flr_{L2R} = 0 \wedge$
$flrd_R = flrfb_R = flr_R = flr_{R2L} = 0] \wedge$
$[rebalctrl = OFF \Rightarrow$
$flr_{L2R} = 0 \wedge flr_{R2L} = 0 \wedge$
$[pumpctrl = BOTH \Rightarrow$
$flrd_L = c_L flrd_E \wedge flrd_R = c_R flrd_E \wedge$
$flr_L = flrd_L + flrfb_L \wedge$
$flr_R = flrd_R + flrfb_R] \wedge$
$[pumpctrl = LEFT \Rightarrow$
$flrd_L = flrd_E \wedge flr_L = flrd_L + flrfb_L \wedge$
$flrd_R = flrfb_R = flr_R = 0] \wedge$
$[pumpctrl = RIGHT \Rightarrow$
$flrd_L = flrfb_L = flr_L = 0 \wedge$
$flrd_R = flrd_E \wedge flr_R = flrd_R + flrfb_R] \,] \wedge$
$[rebalctrl = L2R \Rightarrow$
$[pumpctrl = BOTH \Rightarrow$
$flrd_L = c_L c_{L2R,E} flrd_E \wedge$
$flrd_R = (c_R + c_L c_{L2R,R}) flrd_E \wedge$
$flr_{L2R} = c_L c_{L2R,R} flr_L \wedge$
$flr_{R2L} = 0 \wedge$
$flr_L = flrd_L + flr_{L2R} + flrfb_L \wedge$
$flr_R = flrd_R + flrfb_R] \wedge$
$[pumpctrl = RIGHT \Rightarrow$
$flrd_L = flrfb_L = flr_L = flr_{L2R} = 0 \wedge$
$flrd_R = flrd_E \wedge flr_R = flrd_R + flrfb_R \wedge$
$flr_{R2L} = 0] \,] \wedge$
$[rebalctrl = R2L \Rightarrow$
$[pumpctrl = BOTH \Rightarrow$
$flrd_L = (c_L + c_R c_{R2L,L}) flrd_E \wedge$
$flrd_R = c_R c_{R2L,E} flr_R \wedge$
$flr_{L2R} = 0 \wedge$
$flr_{R2L} = c_R c_{R2L,L} flr_R \wedge$
$flr_L = flrd_L + flrfb_L \wedge$
$flr_R = flrd_R + flr_{R2L} + flrfb_R] \wedge$
$[pumpctrl = LEFT \Rightarrow$
$flrd_L = flrd_E \wedge flr_L = flrd_L + flrfb_L \wedge$
$flrd_R = flrfb_R = flr_R = flr_{R2L} = 0 \wedge$
$flr_{L2R} = 0] \,]$
END
END

Figure 8: Level 2 Hybrid Event-B model of the fuel delivery system. Introduction of the fuel flow rates to the engine.

either pump. Therefore there is no delivered output to the engine either: $\mathit{flr}_L, \mathit{flr}_R, \mathit{flrd}_L, \mathit{flrd}_R, \mathit{flrfb}_L, \mathit{flrfb}_R$ are all 0. Neither can there be any rebalancing going on: flr_{L2R} and flr_{R2L} are also both 0, regardless of the setting of the rebalance control. When flr_E^{CH} is nonzero, this case is interesting, since then, the collection of constraints $0 < \mathit{flrd}_E^{CH} = \mathit{flr}_E = \mathit{flrd}_L + \mathit{flrd}_R = 0 + 0$ is unsatisfiable. The semantics of Hybrid Event-B stipulates that if the specification of a pliant event becomes infeasible (as is the case here), and there is no mode event enabled at that moment, then the execution stops. Here, it corresponds to the case of the pilot switching the pumps off while the aircraft is flying. This causes engine failure and the aircraft crashes (unless the pilot restarts the engine). So this is well represented in our model.

Equally interesting is the case of the engine catching fire. Now, the pilot isolates the fuel supply from the engine to allow the fire to go out: $\mathit{flr}_E^{CH} = 0$. Because the fuel is incompressible, the delivered fuel rates become 0 too, and so any raw pumping outputs drop to zero too (according to cases discussed below), with all pumping output returned to the relevant tank. The pilot can now switch the pumps off while the fire is going out. Once the fire is extinguished, the pilot can switch the pumps on again, and then restart the engine ($\mathit{flr}_E^{CH} > 0$). This sequence of events does not cause infeasibility, so is also well represented in our model.

The remaining cases are easiest to understand according to the setting of *rebalctrl*, starting with the $\mathit{rebalctrl} = \mathit{OFF}$ case. Then, when $\mathit{pumpctrl} = \mathit{BOTH}$, the left and right pumps' delivered output to the engine are respectively proportional to c_L and c_R times their raw output (for the given engine demand, the rest going to bypass). When *pumpctrl* is *LEFT* or *RIGHT* then the relevant pump is solely responsible for its delivered output matching the engine demand.

We examine the case when $\mathit{rebalctrl} = \mathit{L2R}$, noting that the *R2L* case is symmetrical. When $\mathit{pumpctrl} = \mathit{BOTH}$, not only are the output rates scaled by c_L and c_R, but the left pump's c_L-scaled output is further scaled by $c_{L2R,E}$, a relatively small number, to reflect the small contribution that the left pump makes to feeding the engine in this case, since most of its output (the $c_{L2R,R}$ proportion) will be refilling the right tank. To maintain the total demand to the engine, the right pump must supply an additional $c_L\, c_{L2R,R}\, \mathit{flrd}_E$ fuel rate to make up for the left pump's shortfall. (The right pump can obviously do this since it can supply any engine demand by itself anyway.) In the $\mathit{pumpctrl} = \mathit{RIGHT}$ case no refilling takes place since the left pump is inactive — it is like the *OFF*/*LEFT* case above. Finally, the $\mathit{pumpctrl} = \mathit{LEFT}$ case is excluded by the invariants (since it is assumed that $c_L\, c_{L2R,E}\, \mathit{flrd}_E$ is not enough to feed the engine).

We comment further on the nondeterminism of this specification. Consider the $\mathit{pumpctrl} = \mathit{BOTH}$ case without rebalancing. At any moment, the demand $\mathit{flr}_E^{CH} = \mathit{flrd}_E$ is fulfilled by $\mathit{flrd}_L + \mathit{flrd}_R$. This is a combination of two nondeterministic quantities, so may supply the needed value in many ways. In reality, what governs the actual flows of fuel in those parts of the system made accessible by the valve settings, is a combination of: the power of the pumps, the hydrostatic resistances needed to activate the bypass mechanisms in each pump, the relative hydrostatic resistances of the connecting pipework, the hydrostatic resistances in the collector/engine assembly, and the requirement of maintaining a single value of hydrostatic pressure throughout the considered system owing to the incompressible nature of the fuel. Since we do not model these things explicitly, our approach is but an approximation to the reality of such a system, and the nondeterministic (and time dependent) nature of the contributing values makes up for our ignorance of the details. Still, the proportionality factors we use give a reasonable indication of the portion of the pumps' outputs being used in the various cases.

If $\mathit{rebalctrl} = \mathit{L2R}$ this aspect is exacerbated. The fuel demand $\mathit{flr}_E^{CH} = \mathit{flrd}_E$ is now fulfilled by $c_L\, c_{L2R,E}\, \mathit{flrd}_E + (c_R + c_L\, c_{L2R,R})\, \mathit{flrd}_E$, a more complicated combination involving four nondeterministic quantities, with a further expression $c_L\, c_{L2R,R}\, \mathit{flr}_L$ describing the flow to the right tank. Whether this is, in reality, credible as given, with a multiplicative rescaling taking account of the flow distribution, depends again on the factors mentioned. However, the remainder of our development is not critically affected by this, so we retain this style of description for the sake of simplicity.

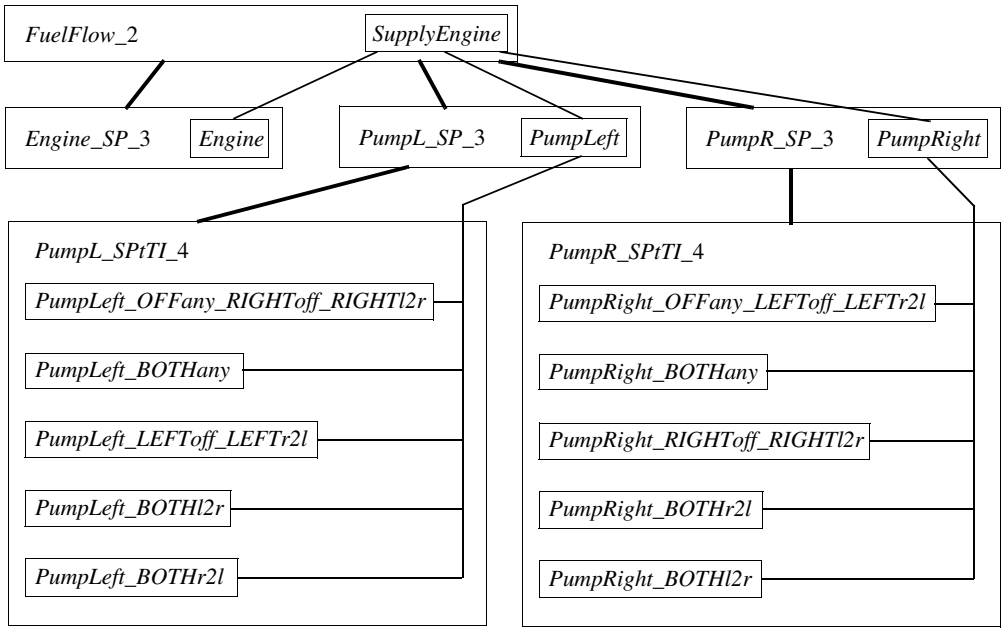

Figure 9: The fate of the *SupplyEngine* pliant event through the system decomposition, when partitioning in space precedes partitioning in time. Machine *FuelFlow_2* is decomposed into *Engine_SP_3*, *PumpL_SP_3*, *PumpR_SP_3*, and the latter two become *PumpL_SPtTI_4*, *PumpR_SPtTI_4*. *SupplyEngine* is decomposed into the events shown inside these machines.

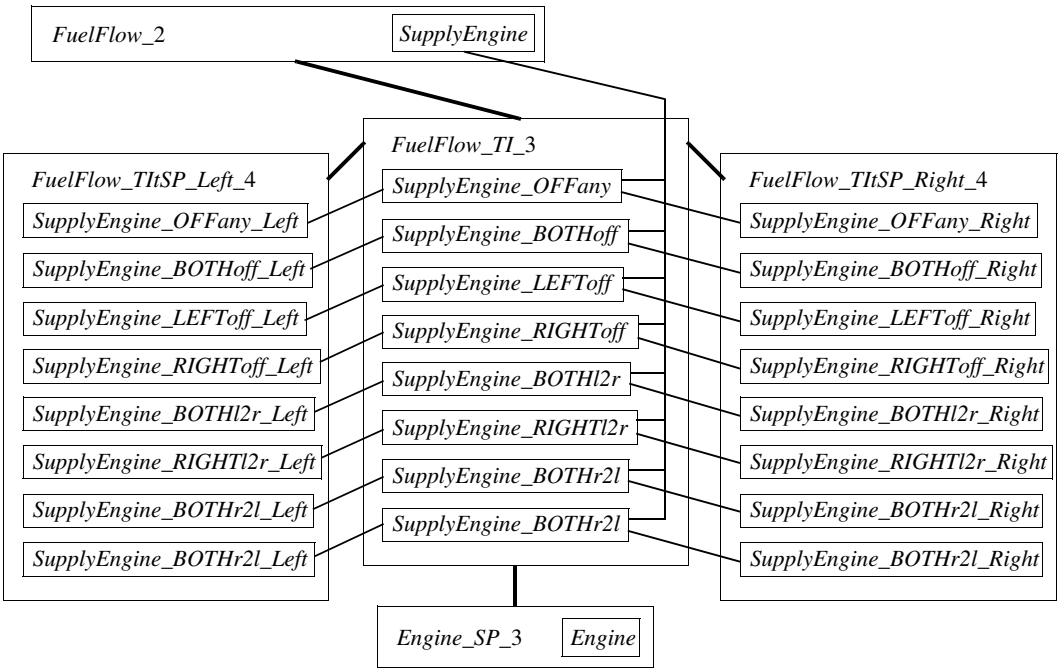

Figure 10: The fate of the *SupplyEngine* pliant event through the system decomposition, when partitioning in time precedes partitioning in space. Machine *FuelFlow_2* is decomposed into *FuelFlow_TI_3* which is further decomposed into *Engine_SP_3*, *FuelFlow_TItSP_Left_4*, *FuelFlow_TItSP_Right_4*. *SupplyEngine* is decomposed into the events shown inside these machines. The greater verbosity of this approach compared with Fig. 9 is evident.

8. Further Refinement and Partitioning

As the development process progresses, the models created get tend to get bigger unless active measures are taken to avoid this. We anticipated this above by including new detail in new machines, on the understanding that the multi-machine version of Hybrid Event-B has been expressly designed to make this both possible and equivalent to a large monolithic refinement step. Taking a large monolithic machine and breaking it up into an equivalent collection of smaller machines can be called *partition in space*.

As a counterpart to this though, we also want to consider *partition in time*. This is a phenomenon akin to decomposing a sequential program into its individual steps, except that it applies to pliant events. The possibility arises rather naturally, since the behaviour of physical equipment is usually governed by local laws, which act largely independently of the context. For example the behaviour of a resistor is given by Ohm's Law, which can be stated independently of the circuit in which the resistor is located. Such a 'global' description can be contrasted with a description of behaviour in particular episodes of time, when the current has some particular value, etc. The latter can be seen as a refinement of the former. (Additionally, it is unlikely that any useful safety invariant can be proved on the basis of Ohm's Law *alone*, whereas with the addition of more detailed application specific constraints, useful safety invariants may become provable.)

A major objective of this paper is to discuss the tradeoffs between partition in space and partition in time in the context of our case study. The partitioning we already did separated the mode events from the pliant events, and for the sake of keeping the modelling within a reasonable size for a paper of this kind, we continue to respect this partitioning in the remainder of this work. Additionally, we only have one nontrivial pliant event to contend with in a separate machine, which is a good starting point, and makes our task manageable. We consider the pros and cons of partitioning in space first followed by partitioning in time, and then the converse, partitioning in time first followed by partitioning in space.

Fig. 9 and Fig. 10 give an illustration of the structure of the two developments, highlighting the plaint events which embody the crucial features that arise, eliminating clutter. Machines are drawn in boxes, with the pliant events within them in smaller boxes. Thick lines show how machines are partitioned, while thinner lines show the event partitions that are subordinate to the machine partitioning. The two developments are discussed in more detail in the following sections.

9. Partitioning in Space First, then Time

Our case study is a bit unusual in that we defined the behaviour of the fuel flow in the pipework via an explicit case analysis rather than a single universally applicable physical law such as Ohm's Law. This meant that we already had to decompose all the behaviours to a lower level of abstraction than would naturally occur in a design process that worked top down. Nevertheless, what we have will serve well enough to illustrate our argument.

In this section we partition in space first, then time. Regarding partition in space, at the opposite extreme of combining everything into a single machine, is having every single physical component (which approximately amounts to each individual variable in our description) in a machine of its own. However, a more reasonable partition would split the system into an engine machine (focusing on engine quantities), and left and right tank machines (focusing on the respective pumps and valves).

Since we are keeping the structure of the external control of the system unchanged, the mode events in the *Pilot* and *Controller* machines remain unchanged. This means that the partition in space has only to focus on the single pliant event *SupplyEngine*, since we can rely on the preemption mechanism to schedule the different cases of *SupplyEngine*, once we have connected all the machines together in the right way.

PROJECT *FuelPump_Prj_SP_3*
REFINES *FuelPump_Prj_2*
INTERFACES
Central_IF_1
FuelFlow_SP_IF_3
MACHINES
Pilot_1
Controller_1
Engine_SP_3
PumpL_SP_3
PumpR_SP_3
SYNCH(*PumpOFF*) ... ...
END

INTERFACE *FuelFlow_SP_IF_3*
PLIANT $flrd_E$
INVARIANTS
$flrd_E \in \mathbb{R} \wedge flrd_E \in \{0\} \cup [ERT_{MIN} \ldots ERT_{MAX}]$
INITIALISATION
BEGIN
$flrd_E := 0$
END
END

MACHINE *Engine_SP_3*
DECOMPOSES *FuelFlow_2*
CONNECTS *FuelFlow_SP_IF_3*
EVENTS
Engine
DECOMPOSES *SupplyEngine*
STATUS pliant
ANY
$flr_E^{CH}, c_L!, c_R!, c_{L2R,E}!, c_{L2R,R}!, c_{R2L,E}!, c_{R2L,L}!$
WHERE
$flr_E^{CH} \in \mathbb{R} \wedge$
$flr_E^{CH} \in \{0\} \cup [ERT_{MIN} \ldots ERT_{MAX}] \wedge$
COMPLY
$flrd_E = flr_E^{CH} \wedge$
$\{c_L!, c_R!, c_{L2R,E}!, c_{L2R,R}!, c_{R2L,E}!, c_{R2L,L}!\} \subset \mathbb{R} \wedge$
$\{c_L!, c_R!, c_{L2R,E}!, c_{L2R,R}!, c_{R2L,E}!, c_{R2L,L}!\} \subset (0 \ldots 1) \wedge$
$c_L! + c_R! = 1 \wedge |c_L! - c_R!| < H \wedge$
$c_{L2R,E}! + c_{L2R,R}! = 1 \wedge c_{L2R,E}!/c_{L2R,R}! < H \wedge$
$c_{R2L,E}! + c_{R2L,L}! = 1 \wedge c_{R2L,E}!/c_{R2L,L}! < H$
END
END

MACHINE *PumpL_SP_3*
DECOMPOSES *FuelFlow_2*
CONNECTS *FuelFlow_SP_IF_3*
READS *Central_IF_1*
PLIANT $flrd_L, flr_L, flrfb_L, flr_{L2R}$
INVARIANTS
$flrd_L \in \mathbb{R} \wedge flrd_L \in \{0\} \cup [PRT_{MIN} \ldots PRT_{MAX}]$
$flr_L \in \mathbb{R} \wedge flr_L \in \{0\} \cup [RPRT_{MIN} \ldots RPRT_{MAX}]$
$flrfb_L \in \mathbb{R} \wedge flrfb_L \in \{0\} \cup [PRT_{MIN} \ldots PRT_{MAX}]$
$flr_{L2R} \in \mathbb{R} \wedge flr_{L2R} \in \{0\} \cup [PRT_{MIN} \ldots PRT_{MAX}]$
EVENTS
INITIALISATION
BEGIN
$flrd_L, flr_L, flrfb_L, flr_{L2R} := 0, 0, 0, 0$
END
PumpLeft
DECOMPOSES *SupplyEngine*
STATUS pliant
ANY $c_L?, c_{L2R,E}?, c_{L2R,R}?$
WHERE $c_L?, c_{L2R,E}?, c_{L2R,R}? \in \mathbb{R}, \mathbb{R}, \mathbb{R}$
COMPLY
$[pumpctrl = OFF \vee$
$(pumpctrl = RIGHT \wedge$
$(rebalctrl = OFF \vee rebalctrl = L2R))$
$\Rightarrow$
$flrd_L = flrfb_L = flr_L = flr_{L2R} = 0] \wedge$
$[pumpctrl = LEFT \wedge$
$(rebalctrl = OFF \vee rebalctrl = R2L)$
$\Rightarrow$
$flrd_L = flrd_E \wedge flr_{L2R} = 0 \wedge$
$flr_L = flrd_L + flrfb_L] \wedge$
$[pumpctrl = BOTH \wedge rebalctrl = OFF \Rightarrow$
$flrd_L = c_L? flrd_E \wedge flr_{L2R} = 0 \wedge$
$flr_L = flrd_L + flrfb_L] \wedge$
$[pumpctrl = BOTH \wedge rebalctrl = L2R \Rightarrow$
$flrd_L = c_L? c_{L2R,E}? flrd_E \wedge$
$flr_{L2R} = c_L? c_{L2R,R}? flr_L \wedge$
$flr_L = flrd_L + flr_{L2R} + flrfb_L] \wedge$
$[pumpctrl = BOTH \wedge rebalctrl = R2L \Rightarrow$
$flrd_L = (c_L? + c_R? c_{R2L,L}?) flrd_E \wedge$
$flr_{L2R} = 0 \wedge flr_L = flrd_L + flrfb_L]$
END
END

Figure 11: Level SP_3 Hybrid Event-B model of the fuel delivery system. Decomposition in space, then time: the space step.

PROJECT *FuelPump_Prj_SPtTI_4*
REFINES *FuelPump_Prj_SP_3*
INTERFACES
Central_IF_1
FuelFlow_SP_IF_3
MACHINES
Pilot_1
Controller_1
Engine_SP_3
PumpL_SPtTI_4
PumpR_SPtTI_4
SYNCH(*PumpOFF*
REFINES *PumpOFF*)
Pilot_1.PumpOFF_S,
Controller_1.PumpOFF_S
PumpL_SPtTI_4.ModeSkip_S
PumpR_SPtTI_4.ModeSkip_S
END
... ...
END

MACHINE *PumpL_SPtTI_4*
REFINES *PumpL_SP_3*
CONNECTS *FuelFlow_SP_IF_3*
READS *Central_IF_1*
PLIANT $flrd_L, flr_L, flrfb_L, flr_{L2R}$
INVARIANTS
$flrd_L \in \mathbb{R} \wedge flrd_L \in \{0\} \cup [PRT_{MIN} \ldots PRT_{MAX}]$
$flr_L \in \mathbb{R} \wedge flr_L \in \{0\} \cup [RPRT_{MIN} \ldots RPRT_{MAX}]$
$flrfb_L \in \mathbb{R} \wedge flrfb_L \in \{0\} \cup [PRT_{MIN} \ldots PRT_{MAX}]$
$flr_{L2R} \in \mathbb{R} \wedge flr_{L2R} \in \{0\} \cup [PRT_{MIN} \ldots PRT_{MAX}]$
EVENTS
INITIALISATION
REFINES *INITIALISATION*
BEGIN
$flrd_L, flr_L, flrfb_L, flr_{L2R} := 0, 0, 0, 0$
END
ModeSkip_S
STATUS convergent
BEGIN
skip
END
... ...

... ...
PumpLeft_OFFany_RIGHToff_RIGHTl2r
REFINES *PumpLeft*
STATUS pliant
ANY $c_L?, c_{L2R,E}?, c_{L2R,R}?$
WHERE $c_L?, c_{L2R,E}?, c_{L2R,R}? \in \mathbb{R}, \mathbb{R}, \mathbb{R} \wedge$
$(pumpctrl = OFF \wedge$
$rebalctrl \in \{OFF, L2R, R2L\}) \vee$
$(pumpctrl = RIGHT \wedge$
$rebalctrl \in \{OFF, L2R\})$
COMPLY
$flrd_L = flrfb_L = flr_L = flr_{L2R} = 0$
END
PumpLeft_BOTHoff
$\ldots pumpctrl = BOTH \wedge rebalctrl = OFF$
COMPLY
$flrd_L = c_L?\, flrd_E \wedge flr_{L2R} = 0 \wedge$
$flr_L = flrd_L + flrfb_L$
END
PumpLeft_LEFToff_LEFTr2l
$\ldots pumpctrl = LEFT \wedge$
$rebalctrl \in \{OFF, R2L\}$
COMPLY
$flrd_L = flrd_E \wedge flr_{L2R} = 0 \wedge$
$flr_L = flrd_L + flrfb_L$
END
PumpLeft_BOTHl2r
$\ldots pumpctrl = BOTH \wedge rebalctrl = L2R$
COMPLY
$flrd_L = c_L?\, c_{L2R,E}?\, flrd_E \wedge$
$flr_{L2R} = c_L?\, c_{L2R,R}?\, flr_L \wedge$
$flr_L = flrd_L + flr_{L2R} + flrfb_L$
END
PumpLeft_BOTHr2l
$\ldots pumpctrl = BOTH \wedge rebalctrl = R2L$
COMPLY
$flrd_L = (c_L? + c_R?\, c_{R2L,L}?)\, flrd_E \wedge$
$flr_{L2R} = 0 \wedge flr_L = flrd_L + flrfb_L$
END
END

Figure 12: Level SPtTI_4 Hybrid Event-B model of the fuel delivery system. Decomposition in space, then time: the time step after the space step.

Fig. 11 shows the space decomposition step of the current design approach. Named the SP_3 level (for space (first) level 3), the *FuelPump_Prj_SP_3* project shows the retained level 1 components, and introduces new machines *Engine_SP_3*, *PumpL_SP_3*, *PumpR_SP_3*, and a new interface *FuelFlow_SP_IF_3*. The latter just holds the variable $flrd_E$ delivered to the engine, since that is the only variable that must be visible to all three machines. The remaining flow variables are the concern of one or other of the pumps alone, so may be assigned to the relevant pump machine. The two pump machines, *PumpL_SP_3* and *PumpR_SP_3*, each connect to the *FuelFlow_SP_IF_3* interface, but need only READ the *Central_IF_1* since there are no new invariants involving the variables in *Central_IF_1* to take into account.

An interesting point concerns how the decomposed system handles the collection of constraints such as $c_L + c_R = 1$ from the WHERE clause of *SupplyEngine* in *FuelFlow_2*. In a single machine, these are simply constraints imposed on a number of locally declared quantities, thus uncontroversial. In the decomposed case, they involve a coupling between values that are now held in separate machines. Here, we use the I/O paradigm of Hybrid Event-B to allow one machine (in this case, the engine machine, *Engine_SP_3*, for symmetry) to regard them as output quantities, $c_L!$ and $c_R!$, and the other machines involved, *PumpL_SP_3* and *PumpR_SP_3* to see them as input quantities, $c_L?$ and $c_R?$. The Hybrid Event-B I/O paradigm regards such variables as bound within the concurrent execution of the pliant events in all three component machines,[5] and thus the I/O becomes an instantaneous private communication stream between a single writer and two readers. Since the details of the communication mechanism are invisible to all other machines (because of the bound variable semantics), the communication is equivalent to a continuous assignment as in the parent event.

One casualty of the decomposition being performed is the clause $flrd_E = flrd_L + flrd_R$ from the *SupplyEngine* event in the *FuelPump_Prj_2* project. With a more complicated variable sharing strategy, this could have been included, but in order to save space, we pursued a variable sharing strategy that made this impossible. Still, we have not lost anything, since the fact in question is a consequence of other facts in the development, specifically of (the modified presentation of) $c_L + c_R = 1$ and its bretheren.

Regarding the pump behaviours in the pliant event *SupplyEngine* of Fig. 8, the various behaviours of the two pumps become separated in the decomposition. There are five of them per pump, and they are shown in the pliant event *PumpLeft* of *PumpL_SP_3*. Observe that because of the low level case analysis in the parent event *SupplyEngine*, the different individual pump behaviours in *SupplyEngine* had to be aggregated into the cases of *PumpLeft*. In a top down development in which natural physical laws predominated in the pliant behaviours, this kind of need to aggregate would be less pronounced. The right pump, *PumpRight* in machine *PumpR_SP_3* is a mirror image of the left pump case, and is not shown in Fig. 11.

As in the undecomposed version of *SupplyEngine*, when the machines ran, the relevant cases from each pliant event would be selected, moment by moment, according to the external demand and control settings. In the case of Fig. 8, the *SupplyEngine* event is always enabled, and is restarted according to the correct case, after every mode event preemption at runtime. In the partitioned case, the same thing happens, but the mode event occurrences are synchronised across the component machines, and the selection of the correct cases in the pump pliant events takes place independently in each component machine.

Fig. 12 shows what happens when we follow the partition in space by a partition in time. The project is *FuelPump_Prj_SPtTI_4* (space then time, level 4). The main thing that happens is that the single pliant event in each level 3 component machine gets split into its five cases, each now becoming an event in its

[5]The DECOMPOSES keyword relating the component events to their parent event *SupplyEngine*, guarantees that all the needed events will indeed execute concurrently.

own right. To ensure the right event gets scheduled only at the right time, each of the new events has to have its own guard, constructed from the enabling clause in the parent event. Additionally, each event has to have its own name, and these are also constructed from the enabling conditions in the parent event. Thus *PumpLeft_OFFany_RIGHToff_RIGHTl2r*, the pump inactive case, is named after the clauses of its guard: $pumpctrl = OFF$ (and *rebalctrl* irrelevant) or $pumpctrl = RIGHT \wedge rebalctrl \in \{OFF, L2R\}$.

Being events in their own right, each now needs a fully fledged appropriate header. This makes their description very verbose compared with the earlier version. We show in full only the header for *PumpLeft_OFFany_RIGHToff_RIGHTl2r*. As well as the guard that we have mentioned, the header needs to declare its refining event, its status, and the local input variables needed for the relevant $c_?$ values. For the other four events, we indicate only the guard terms, the rest of the header being identical.

Since there are now separate pliant events in the each of the two pump machines (rather than a single pliant event reacting over time to changing conditions in each machine), their occurrences need to be interleaved with suitable mode event occurrences. For this purpose a mode event *ModeSkip_S* has been introduced into the *PumpL_SPtTI_4* machine with a similar event in for the right pump. As is easily seen, these events just skip, but they help ensure that the scheduling rules in Sections 3.1 and 3.2 are obeyed. To ensure they are obeyed, their occurrences must be synchronised with the pilot's commands, so the earlier synchronisations are enlarged to include occurrences of the two pump machines' *ModeSkip_S* events. Fig. 12 just shows the enlarged *PumpOFF* synchronisation.

10. Partitioning in Time First, then Space

In this section we consider the alternative approach: partitioning in time first, followed by partitioning in space. We start again with Fig. 8, and decompose it a different way. Fig. 13 shows the time decomposition. The project is now *FuelPump_Prj_TI_3* (time (first) level 3). The general idea follows the pattern described in the last step of the space-then-time approach, but this time there is only one machine and a single plaint event. Thus, the most notable effect that can be seen is the increased verbosity due to the replicated headers of all the constituent events. Since no decomposition in space has taken place yet, the full header of the *FuelFlow_2* machine's *SupplyEngine* event is replicated (with the only variation being the enabledness guard) in all the individual events of the decomposed machine. Again, we only indicate the enabledness guard in each event to save space. Even then, the larger bodies of the decomposed events mean that there was not enough room to show the last two events, *SupplyEngine_BOTHr2l* and *SupplyEngine_LEFTr2l*, in the figure. They are mirror images of the *SupplyEngine_BOTHl2r* and *SupplyEngine_RIGHTl2r* events just above. As before, because we now have separate pliant events, we need a synchronised skip mode event to interleave them, but only one, since there is still only one machine involved in the decomposition. This affects the synchronisations in the *FuelPump_Prj_TI_3* project in the expected way.

Doing the partition in space now, Fig. 14 shows the result, which is as expected. The many individual events of the *FuelPump_Prj_TI_3* project's *FuelFlow_TI_3* machine get split into their left pump and right pump parts. Fig. 14 shows only the left pump machine and its events. Given the number of events in the *FuelFlow_TI_3* machine, and given that we strove to make both approaches as compatible as possible, it is not surprising that the number of events in a pump machine derived in the time-then-space approach exceeds the number derived in the space-then-time approach. Clearly the space-then-time approach is to be preferred for the sake of economy. We comment further below.

11. The Approaches Compared, and a General Strategy

The mischievous reader might well complain at this point that the number of events in a pump machine derived in the time-then-space approach does not *really* exceed the number derived in the space-then-time approach, at least not if we had neglected to aggregate the pump behaviours in the space step

PROJECT *FuelPump_Prj_TI_3*
REFINES *FuelPump_Prj_2*
INTERFACES
Central_IF_1
MACHINES
Pilot_1
Controller_1
FuelFlow_TI_3
... ...

... ...
SYNCH(*PumpOFF*
REFINES *PumpOFF*)
Pilot_1.PumpOFF_S,
Controller_1.PumpOFF_S
FuelFlow_TI_3.ModeSkip_S
END
... ...
END

MACHINE *FuelFlow_TI_3*
REFINES *FuelFlow_2*
READS *Central_IF_1*
PLIANT $flrd_E, flrd_L, flrd_R, flr_L, flr_R, flrfb_L, flrfb_R, flr_{L2R}, flr_{R2L}$
INVARIANTS
$flrd_E \in \mathbb{R} \wedge flrd_E \in \{0\} \cup [ERT_{MIN} \ldots ERT_{MAX}]$
$flrd_L \in \mathbb{R} \wedge flrd_L \in \{0\} \cup [PRT_{MIN} \ldots PRT_{MAX}]$
$flrd_R \in \mathbb{R} \wedge flrd_R \in \{0\} \cup [PRT_{MIN} \ldots PRT_{MAX}]$
$flr_L \in \mathbb{R} \wedge flr_L \in \{0\} \cup [RPRT_{MIN} \ldots RPRT_{MAX}]$
$flr_R \in \mathbb{R} \wedge flr_R \in \{0\} \cup [RPRT_{MIN} \ldots RPRT_{MAX}]$
$flrfb_L \in \mathbb{R} \wedge flrfb_L \in \{0\} \cup [PRT_{MIN} \ldots PRT_{MAX}]$
$flrfb_R \in \mathbb{R} \wedge flrfb_R \in \{0\} \cup [PRT_{MIN} \ldots PRT_{MAX}]$
$flr_{L2R} \in \mathbb{R} \wedge flr_{L2R} \in \{0\} \cup [PRT_{MIN} \ldots PRT_{MAX}]$
$flr_{R2L} \in \mathbb{R} \wedge flr_{R2L} \in \{0\} \cup [PRT_{MIN} \ldots PRT_{MAX}]$
EVENTS
INITIALISATION
REFINES *INITIALISATION*
BEGIN
$flrd_E, flrd_L, flrd_R, flr_L, flr_R, flrfb_L, flrfb_R, flr_{L2R}, flr_{R2L} := 0,0,0,0,0,0,0,0,0$
END
ModeSkip_S
STATUS convergent
BEGIN skip END
SupplyEngine_OFFany
REFINES *SupplyEngine*
STATUS pliant
ANY $flr_E^{CH}, c_L, c_R, c_{L2R,E}, c_{L2R,R}, c_{R2L,E}, c_{R2L,L}$
WHERE
$flr_E^{CH} \in \mathbb{R} \wedge$
$flr_E^{CH} \in \{0\} \cup [ERT_{MIN} \ldots ERT_{MAX}] \wedge$
$\{c_L, c_R, c_{L2R,E}, c_{L2R,R}, c_{R2L,E}, c_{R2L,L}\} \subset \mathbb{R} \wedge$
$\{c_L, c_R, c_{L2R,E}, c_{L2R,R}, c_{R2L,E}, c_{R2L,L}\} \subset (0 \ldots 1) \wedge$
$c_L + c_R = 1 \wedge |c_L - c_R| < H \wedge$
$c_{L2R,E} + c_{L2R,R} = 1 \wedge c_{L2R,E}/c_{L2R,R} < H \wedge$
$c_{R2L,E} + c_{R2L,L} = 1 \wedge c_{R2L,E}/c_{R2L,L} < H \wedge$
$pumpctrl = OFF$
COMPLY
$flrd_E = flr_E^{CH} \wedge flrd_E = flrd_L + flrd_R \wedge$
$flrd_L = flrfb_L = flr_L = flr_{L2R} = 0 \wedge$
$flrd_R = flrfb_R = flr_R = flr_{R2L} = 0$
END
... ...

... ...
SupplyEngine_BOTHoff
$\ldots pumpctrl = BOTH \wedge rebalctrl = OFF$
COMPLY
$flrd_E = flr_E^{CH} \wedge flrd_E = flrd_L + flrd_R \wedge$
$flrd_L = c_L flrd_E \wedge flrd_R = c_R flrd_E \wedge$
$flr_L = flrd_L + flrfb_L \wedge flr_R = flrd_R + flrfb_R \wedge$
$flr_{L2R} = 0 \wedge flr_{R2L} = 0$
END
SupplyEngine_LEFToff
$\ldots pumpctrl = LEFT \wedge rebalctrl = OFF$
COMPLY
$flrd_E = flr_E^{CH} \wedge flrd_E = flrd_L + flrd_R \wedge$
$flrd_L = flrd_E \wedge flr_L = flrd_L + flrfb_L \wedge$
$flrd_R = flrfb_R = flr_R = flr_{R2L} = 0 \wedge$
$flr_{L2R} = 0$
END
SupplyEngine_RIGHToff
$\ldots pumpctrl = RIGHT \wedge rebalctrl = OFF$
COMPLY
$flrd_E = flr_E^{CH} \wedge flrd_E = flrd_L + flrd_R \wedge$
$flrd_L = flrfb_L = flr_L = flr_{L2R} = 0 \wedge$
$flrd_R = flrd_E \wedge flr_R = flrd_R + flrfb_R \wedge$
$flr_{R2L} = 0$
END
SupplyEngine_BOTHl2r
$\ldots pumpctrl = BOTH \wedge rebalctrl = L2R$
COMPLY
$flrd_E = flr_E^{CH} \wedge flrd_E = flrd_L + flrd_R \wedge$
$flrd_L = c_L c_{L2R,E} flrd_E \wedge$
$flrd_R = (c_R + c_L c_{L2R,R}) flrd_E \wedge$
$flr_{L2R} = c_L c_{L2R,R} flr_L \wedge flr_{R2L} = 0 \wedge$
$flr_L = flrd_L + flr_{L2R} + flrfb_L \wedge$
$flr_R = flrd_R + flrfb_R$
END
SupplyEngine_RIGHTl2r
$\ldots pumpctrl = RIGHT \wedge rebalctrl = L2R$
COMPLY
$flrd_E = flr_E^{CH} \wedge flrd_E = flrd_L + flrd_R \wedge$
$flrd_L = flrfb_L = flr_L = flr_{L2R} = 0 \wedge$
$flrd_R = flrd_E \wedge flr_R = flrd_R + flrfb_R \wedge$
$flr_{R2L} = 0$
END
SupplyEngine_BOTHr2l ... ...
SupplyEngine_LEFTr2l ... ...
END

Figure 13: Level TI_3 Hybrid Event-B model of the fuel delivery system. Decomposition in time then space: the time step.

PROJECT *FuelPump_Prj_TItSP_4*
REFINES *FuelPump_Prj_TI_3*
INTERFACES
Central_IF_1
FuelFlow_SP_IF_3
MACHINES
Pilot_1
Controller_1
Engine_SP_3
FuelFlow_TItSP_Left_4
FuelFlow_TItSP_Right_4
... ...

... ...
SYNCH(*PumpOFF*
REFINES *PumpOFF*)
Pilot_1.PumpOFF_S,
Controller_1.PumpOFF_S
FuelFlow_TItSP_Left_4.ModeSkip_S
FuelFlow_TItSP_Right_4.ModeSkip_S
END
... ...
END

MACHINE *FuelFlow_TItSP_Left_4*
DECOMPOSES *FuelFlow_TI_3*
CONNECTS *FuelFlow_SP_IF_3*
READS *Central_IF_1*
PLIANT $flrd_L, flr_L, flrfb_L, flr_{L2R}$
INVARIANTS
$flrd_L \in \mathbb{R} \wedge flrd_L \in \{0\} \cup [PRT_{MIN} \ldots PRT_{MAX}]$
$flr_L \in \mathbb{R} \wedge flr_L \in \{0\} \cup [RPRT_{MIN} \ldots RPRT_{MAX}]$
$flrfb_L \in \mathbb{R} \wedge flrfb_L \in \{0\} \cup [PRT_{MIN} \ldots PRT_{MAX}]$
$flr_{L2R} \in \mathbb{R} \wedge flr_{L2R} \in \{0\} \cup [PRT_{MIN} \ldots PRT_{MAX}]$
EVENTS
INITIALISATION
REFINES *INITIALISATION*
BEGIN
$flrd_L, flr_L, flrfb_L, flr_{L2R} := 0, 0, 0, 0$
END
ModeSkip_S
STATUS convergent
BEGIN skip END
SupplyEngine_OFFany_Left
DECOMPOSES *SupplyEngine_OFFany*
STATUS pliant
ANY $c_L?, c_{L2R,E}?, c_{L2R,R}?$
WHERE
$c_L?, c_{L2R,E}?, c_{L2R,R}? \in \mathbb{R}, \mathbb{R}, \mathbb{R} \wedge$
$pumpctrl = OFF$
COMPLY
$flrd_L = flrfb_L = flr_L = flr_{L2R} = 0$
END
SupplyEngine_BOTHoff_Left
$\ldots pumpctrl = BOTH \wedge rebalctrl = OFF$
COMPLY
$flrd_L = c_L\, flrd_E \wedge flr_L = flrd_L + flrfb_L \wedge$
$flr_{L2R} = 0$
END
... ...

... ...
SupplyEngine_LEFToff_Left
$\ldots pumpctrl = LEFT \wedge rebalctrl = OFF$
COMPLY
$flrd_L = flrd_E \wedge flr_L = flrd_L + flrfb_L \wedge$
$flr_{L2R} = 0$
END
SupplyEngine_RIGHToff_Left
$\ldots pumpctrl = RIGHT \wedge rebalctrl = OFF$
COMPLY
$flrd_L = flrfb_L = flr_L = flr_{L2R} = 0$
END
SupplyEngine_BOTHl2r_Left
$\ldots pumpctrl = BOTH \wedge rebalctrl = L2R$
COMPLY
$flrd_L = c_L\, c_{L2R,E}\, flrd_E \wedge$
$flr_{L2R} = c_L\, c_{L2R,R}\, flr_L \wedge$
$flr_L = flrd_L + flr_{L2R} + flrfb_L$
END
SupplyEngine_RIGHTl2r_Left
$\ldots pumpctrl = RIGHT \wedge rebalctrl = L2R$
COMPLY
$flrd_L = flrfb_L = flr_L = flr_{L2R} = 0$
END
SupplyEngine_BOTHr2l_Left ... ...
SupplyEngine_LEFTr2l_Left ... ...
END

Figure 14: Level TItSP_4 Hybrid Event-B model of the fuel delivery system. Decomposition in time then space: the space step following the time step.

of the space-then-time approach, machine *PumpL_SP_3*, event *PumpLeft*. Indeed this is true. Without the aggregation, and given the intentional compatibility of the two approaches, we would have arrived at exactly the same result, modulo renaming. What has our comparison achieved then?

The answer depends on a range of factors. The main one would appear to be the extent to which, in a top down development strategy, pliant behaviours are most naturally given in terms of universal physical laws (e.g. the Ohm's Law example give earlier), or whether they arise via a more detailed case analysis. In the former case, the kind of aggregation we encountered would not naturally be needed, since the more detailed descriptions of the pliant behaviour would not have been given yet at an early enough stage of development. In the latter case, some aggregation might well be needed, if the system was described via cases.

Nevertheless, even in our example development, if we wished to aggregate at the last stage of the time-then-space approach, to eliminate the redundancy of pump events which had semantically identical bodies (this being in order to achieve the same granularity of description as in the space-then-time approach), the task would be rendered more difficult because the preceding time decomposition introduced naming conventions for the various events that obscured their essential identity. This is a general phenomenon that would be present in any such development, and indicates that the space-then-time approach offers some technical advantages, especially when it is to be supported by mechanisation, for which inferring the needed semantic identity could prove challenging. In general, it is preferable to avoid the potential combinatorial explosion that the generation of redundant events would create event if the identity detection problem could be overcome by mechanisation.

Therefore, building on the insights gained in earlier Hybrid Event-B development case studies and enhancing them with the lessons learned here, we can recommend a Hybrid Event-B development strategy as follows.

- Start by developing the mode view of the system. Use a default pliant event such a *PliTrue* to ensure correct semantics. Restrict refinements to mode events only, until it becomes appropriate to introduce nontrivial pliant behaviour.
- Introduce additional decomposition and synchronisation into the system model as needed, to permit each self-contained piece of physical apparatus to be contained in a machine of its own.
- Introduce the required nontrivial pliant behaviours into the various machines of the system, profiting from the decomposition to avoid the risk of combinatorial explosion in the pliant events.
- Continue to refine until the desired level of detail has been achieved.

The above constitutes the culmination of our deliberations about the optimal approach to the use of the structuring facilities of Hybrid Event-B.

12. The Fuel System Revisited

We are now in a position to comment on the fuel system case study in the light of the enriched decomposition framework of Section 5. Firstly, it is clear that the elaboration of mode event decomposition in Section 5 makes no impact, since during the decomposition steps, the relevant machines only had trivial mode events. However, the decomposition of pliant events in the case study conforms to the scheme described in Section 5, in the following manner.

Firstly, the loss of $\mathit{flrd}_E = \mathit{flrd}_L + \mathit{flrd}_R$ noted in Section 9 is covered by the approach in Section 5. As argued in Section 9, the conjunction of the COMPLY clauses in the events that the decomposition step produced is equivalent to the originating COMPLY clause.

Secondly, in each machine that is decomposed in space, the comply clauses of the pliant event(s) in question can easily be manipulated into the form $\bigwedge_i \mathrm{A}_i \Rightarrow \mathrm{B}_i$, where the A_i are mutually exclusive and the B_i are also mutually exclusive. Taking into account all the mutual exclusivity properties of all the various events, and also the model invariants that hold, implies that the form $\bigwedge_i \mathrm{A}_i \Rightarrow \mathrm{B}_i$ can be rewritten to $\bigoplus_i \mathrm{A}_i \wedge \mathrm{B}_i$, which is the form required for the POs (5) and (6) to be applicable. The detailed discussion of the case study in earlier sections now allows us to conclude that the decompositions performed can be justified on formal grounds according to the scheme described above.

13. Related Work

Hybrid Event-B is similar to a number of hybrid systems formulations in the literature. These have been studied intensively for many years, and the literature by now is large. Some of the earliest work includes [39, 6, 7, 32, 37], followed shortly afterwards by works such as [38, 27, 28, 49] and [30, 43, 25, 11]. Slightly later formulations include [37, 21, 33, 34, 24, 8, 23, 26]. Particular noteworthy is [22], which surveys a large number of these formulations and the tools that support them. A more contemporary overview of many of these approaches is [45]. A large body of work has appeared in the *International Conference on Hybrid Systems: Computation and Control* series of international conferences, and nowadays we have the *CPS Week* major annual meeting.

These aside, closest to our approach is the KeYmaera tool described in [41] which also bases its reasoning on properties of symbolic representations of hybrid systems. While the KeYmaera tool accepts relatively arbitrary programs of discrete and continuous transitions, and reasons about them using an appropriate dynamic logic, the Hybrid Event-B approach is focused on safety properties, and its suite of proof obligations is designed to inhibit (as far as possible) the writing of models which are unphysical. (For example successive instantaneous transitions taking place at the same time point are prevented.)

Most approaches to hybrid and cyberphysical systems can be located in a spectrum that ranges from simulation to verification. Simulation is the easiest and most comprehensive approach since it is based on widely applicable algorithms, while verification suffers from the fact that any language that is expressive enough to encompass a significant portion of hybrid behaviour is highly undecidable. Since decidability is highly prized in the verification community, verification based approaches pay the price by severely curtailing expressivity. Even then, the needed decision procedures often have high complexity, adding yet more difficulties.

For approaches more heavily slanted towards simulation, their semantics can be problematic. While the discrete side of the approach is invariably captured precisely enough, the side of the formalism that deals with the continuous side is often treated in much less depth. The standard texts [5, 36] give some indication of this. Typically the continuous semantics is either: precise but severely curtailed in expressivity; or is more encompassing regarding the admitted continuous behaviour but significantly less precise regarding its foundations — in extreme cases delegating *all* aspects of continuous behaviour to, e.g., the semantics of a simulation tool.[6] This puts effective and dependable reasoning about the behaviours that can be described out of reach.

The extent to which any of these characteristics is present in any given formalism varies widely, of course. Our own approach for Hybrid Event-B attempts to bypass some of these difficulties by advocating a top down methodology. By starting with simple models, and designing the properties that they should satisfy along with them (rather than trying to discover those *post hoc*), and enriching both along the way to the final system, the aim is to keep the tractability of all aspects of design and verification much

[6] In fact, the behaviour of commercial simulation tools for physical modelling is often highly customer-driven, and makes no real contact with any foundational semantic concerns [40].

higher than if one was confronted with the final system outright — without any clues as to its underlying structure or design motivations.

14. Conclusions

In this paper we started by outlining a simplified fuel supply system for a small aircraft. This is a system which contains a preponderance of physical apparatus, which was useful in that it provided a good vehicle for the issues that we wanted discuss, specifically the manipulation of nontrivial pliant behaviour through refinement and decomposition. We gave an overview of the essential elements of Hybrid Event-B [18, 19], including what is critical for multi-machine developments, and extended the decomposition approach of [19] to give greater flexibility in practice, which we used later. Then we started to develop the system according to the strategy of attending to the mode event structure first, a strategy that has already proved useful previously. We developed the system to the point where the pliant behaviour of the pump system needed to be brought into the models. We did this in a manner that allowed for fairly straightforward modelling, so as not to obscure the structural issues that were the main focus of our attention here, and that allowed for straightforward manual confirmation of the verification conditions.

Our modelling was based on a case analysis of the pliant behaviour, which had the merit of bringing into the model all the detail needed later, at a relatively early stage of the work. This set the scene for an exploration of decomposition approaches referred to as partition in space and partition in time, illustrated for the principal events affected in Fig. 9 and Fig. 10. We pursued the two alternative orders of partition in order to elucidate their pros and cons. After exploring these, we came up with a standardised general strategy for doing developments of such complex systems in Hybrid Event-B, which we described in Section 11. This offers concrete recommendations for making best use of the technical devices made available in the Hybrid Event-B formalism, particularly in the multi-machine case. Again, in this part of the work, the aim was to keep the verification task to a minimum, easily confirmed by inspection, by requiring the pliant behaviour to only satisfy simple implicit conditions.

Aside from the beneficial structural insights just indicated, the case study work in this paper constitutes a significant exercise in use of the multi-machine facilities of Hybrid Event-B within the confines of a single self-contained account, being neither too big to describe in reasonable detail nor too small to be uninterestingly trivial, nor too challenging to verify by inspection. This confirms the utility and appropriateness of the framework.

References

[1] Summit Report: Cyber-Physical Systems, 2008.
http://iccps2012.cse.wustl.edu/_doc/CPS_Summit_Report.pdf.

[2] J.R. Abrial, The B-Book: Assigning Programs to Meanings, Cambridge University Press, 1996.

[3] J.R. Abrial, Modeling in Event-B: System and Software Engineering, Cambridge University Press, 2010.

[4] J.R. Abrial, M. Butler, S. Hallerstede, T. Hoang, F. Mehta, L. Voisin, Rodin: An Open Toolset for Modelling and Reasoning in Event-B, Int. J. Soft. Tools for Tech. Trans. 12 (2010) 447–466.

[5] R. Alur, Principles of Cyberphysical Systems, MIT Press, 2015.

[6] R. Alur, C. Courcoubetis, T. Henzinger, P.H. Ho, Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems, in: Proc. Workshop on Theory of Hybrid Systems, volume 736 of *LNCS*, Springer, 1993, pp. 209–229.

[7] R. Alur, D. Dill, A Theory of Timed Automata, Theor. Comp. Sci. 126 (1994) 183–235.

[8] G. Audemard, M. Bozzano, A. Cimatti, R. Sebastiani, Verifying Industrial Hybrid Systems with MathSAT, in: Proc. BMC-04, volume 119, ENTCS, 2005, pp. 17–32.

[9] R. Back, K. Sere, Stepwise Refinement of Action Systems, in: Proc. MPC-89, volume 376, Springer, LNCS, 1989, pp. 115–138.

[10] R. Back, J. von Wright, Trace Refinement of Action Systems, in: Proc. CONCUR-94, volume 836, Springer, LNCS, 1994, pp. 367–384.

[11] R.J. Back, L. Petre, I. Porres, Continuous Action Systems as a Model for Hybrid Systems, Nordic J. Comp. 8 (2001) 2–21.

[12] R.J.R. Back, J. von Wright, Refinement Calculus: A Systematic Introduction, Springer, 1998.

[13] R. Banach, Pliant Modalities in Hybrid Event-B, in: Liu, Woodcock, Zhu (Eds.), Proc. Jifeng He Festschrift 2013, volume 8051 of *LNCS*, Springer, 2013, pp. 37–53.

[14] R. Banach, Formal Refinement and Partitioning of a Fuel Pump System for Small Aircraft in Hybrid Event-B, in: Bonsangue, Deng (Eds.), Proc. IEEE TASE-16, IEEE, 2016, pp. 65–72.

[15] R. Banach, The Landing Gear System in Multi-Machine Hybrid Event-B, Int. J. Soft. Tools for Tech. Trans. 19 (2017) 205–228.

[16] R. Banach, M. Butler, A Hybrid Event-B Study of Lane Centering, in: Aiguier, Boulanger, Krob, Marchal (Eds.), Proc. CSDM-13, Springer, 2013, pp. 97–111.

[17] R. Banach, M. Butler, Cruise Control in Hybrid Event-B, in: Liu, Woodcock, Zhu (Eds.), Proc. ICTAC-13, volume 8049 of *LNCS*, Springer, 2013, pp. 76–93.

[18] R. Banach, M. Butler, S. Qin, N. Verma, H. Zhu, Core Hybrid Event-B I: Single Hybrid Event-B Machines, Sci. Comp. Prog. 105 (2015) 92–123.

[19] R. Banach, M. Butler, S. Qin, H. Zhu, Core Hybrid Event-B II: Multiple Cooperating Hybrid Event-B Machines, Sci. Comp. Prog. 139 (2017) 1–35.

[20] L. Barolli, M. Takizawa, F. Hussain, Special Issue on Emerging Trends in Cyber-Physical Systems, J. Amb. Intel. Hum. Comp. 2 (2011) 249–250.

[21] Bender, K. and Broy, M. and Péter, I. and Pretschner, A. and Stauner, T., Model Based Development of Hybrid Systems: Specification, Simulation, Test Case Generation, in: Modelling, Analysis, and Design of Hybrid Systems, volume 279, Springer, LNCIS, 2002, pp. 37–51.

[22] L. Carloni, R. Passerone, A. Pinto, A. Sangiovanni-Vincentelli, Languages and Tools for Hybrid Systems Design, Foundations and Trends in Electronic Design Automation 1 (2006) 1–193.

[23] A. Cimatti, M. Roveri, Requirements Validation for Hybrid Systems, in: Proc. CAV-09, volume 5643, Springer, LNCS, 2009, pp. 188–203.

[24] E. Clarke, A. Fehnker, Z. Han, B. Krogh, O. Stursberg, M. Theobald, Verification of Hybrid Systems Based on Counterexample-Guided Abstraction Refinement, in: Proc. TACAS-03, volume 2619, Springer, LNCS, 2003, pp. 192–207.

[25] A. Deshpande, A. Göllü, P. Varaiya, SHIFT: A Formalism and a Programming Language for Dynamic Networks of Hybrid Automata, in: Proc. Hybrid Systems IV, volume 1273, Springer, LNCS, 1997, pp. 113–133.

[26] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, O. Maler, SpaceEx: Scalable Verification of Hybrid Systems, in: Proc. CAV-11, volume 6806, Springer, LNCS, 2011, pp. 379–395.

[27] V. Friesen, A. Nordwig, M. Weber, Object-Oriented Specification of Hybrid Systems using UML, h and ZimOO, in: Proc. ZUM-98, volume 1493, Springer, LNCS, 1998, pp. 328–346.

[28] V. Friesen, A. Nordwig, M. Weber, Toward an Object-Oriented Design Methodology for Hybrid Systems, Object-Oriented Technology and Computing Systems Re-Engineering (1999) 1.

[29] E. Geisberger, M. Broy (eds.), Living in a Networked World. Integrated Research Agenda Cyber-Physical Systems (agendaCPS), 2015. http://www.acatech.de/fileadmin/user_upload/Baumstruktur_nach_Website/Acatech/root/de/Publikationen/Projektberichte/acaetch_STUDIE_agendaCPS_eng_WEB.pdf.

[30] Grosu, R. and Stauner, T. and Broy, M., A Modular Visual Model for Hybrid Systems, in: Proc. FTRTFT-98, volume 1486, Springer, LNCS, 1998, pp. 75–91.

[31] S. Hallerstede, T. Hoang, Refinement by Interface Instantiation, in: Derrick, Fitzgerald, Gnesi, Khurshid, Leuschel, Reeves, Riccobene (Ed.), Proc. ABZ-12, volume 7316, Springer, LNCS, 2012, pp. 223–237.

[32] J. He, From CSP to Hybrid Systems, in: Roscoe (Ed.), A Classical Mind, Essays in Honour of C.A.R. Hoare, Prentice-Hall, 1994, pp. 171–189.

[33] T. Henzinger, The Theory of Hybrid Automata, in: Proc. IEEE LICS-96, IEEE, 1996, pp. 278–292. Also http://mtc.epfl.ch/~tah/Publications/the_theory_of_hybrid_automata.pdf.

[34] Y. Kesten, Z. Manna, A. Pnueli, Verification of Clocked and Hybrid Systems, Acta Informatica 36 (2000) 837–912.

[35] T. Lecomte, Atelier B has Turned 20, in: Proc. ABZ-16, volume 9675, Springer, LNCS, 2016, p. XVI.

[36] E. Lee, S. Shesha, Introduction to Embedded Systems: A Cyberphysical Systems Approach, LeeShesha.org, 2nd. edition, 2015.

[37] N. Lynch, R. Segala, F. Vaandrager, Hybrid I/O Automata, Information and Computation 185 (2003) 105–157. MIT Technical Report MIT-LCS-TR-827d.

[38] N. Lynch, R. Segala, F. Vaandrager, H. Weinberg, Hybrid I/O Automata, Springer, 1996.

[39] Z. Manna, A. Pnueli, Verifying Hybrid Systems, in: Hybrid Systems, volume 736, Springer, LNCS, 1993, pp. 4–35.

[40] P. Mosterman. Private Communication.

[41] A. Platzer, Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics, Springer, 2010.

[42] Rodin Tool. http://www.event-b.org/ http://www.rodintools.org/ http://sourceforge.net/projects/rodin-b-sharp/.

[43] T. Stauner, B. Rumpe, P. Scholz, Hybrid System Model, 1999. TUM-I9903.

[44] J. Sztipanovits, Model Integration and Cyber Physical Systems: A Semantics Perspective, in: Butler, Schulte (Eds.), Proc. FM-11, Springer, LNCS 6664, p.1, http://sites.lero.ie/download.aspx?f=Sztipanovits-Keynote.pdf, 2011. Invited talk, FM 2011, Limerick, Ireland.

[45] P. Tabuada, Verification and Control of Hybrid Systems: A Symbolic Approach, Springer, 2009.

[46] U.S. Department of Transportation, Federal Aviation Administration, Flight Standards Service, Aviation Maintenance Technician Handbook — Airframe, 2012. http://www.faa.gov/regulations_policies/handbooks_manuals/aircraft/amt_ airframe_handbook/.

[47] L. Voisin, J.R. Abrial, The Rodin Platform has Turned Ten, in: Ait Ameur, Schewe (Eds.), Proc. ABZ-14, volume 8477 of *LNCS*, Springer, 2014, pp. 1–8.

[48] J. Willems, Open Dynamical Systems: Their Aims and their Origins. Ruberti Lecture, Rome, 2007. http://homes.esat.kuleuven.be/~jwillems/Lectures/2007/Rubertilecture.pdf.

[49] C. Zhou, J. Wang, A. Ravn, A Formal Description of Hybrid Systems, in: Alur, Sontag, Henzinger (Eds.), Proc. HS-95, volume 1066, Springer, LNCS, 1995, pp. 511–530.