The composition of Event-B models

Michael Poppleton

School of Electronics and Computer Science,
University of Southampton, Highfield,
Southampton SO17 1BJ, UK,
mrp@ecs.soton.ac.uk

Abstract. The transition from classical B [2] to the Event-B language and method [3] has seen the removal of some forms of model structuring and composition, with the intention of reinventing them in future. This work contributes to that reinvention. Inspired by a proposed method for state-based decomposition and refinement [5] of an Event-B model, we propose a familiar parallel event composition (over disjoint state variable lists), and the less familiar event fusion (over intersecting state variable lists). A brief motivation is provided for these and other forms of composition of models, in terms of feature-based modelling. We show that model consistency is preserved under such compositions. More significantly we show that model composition preserves refinement.

1 Introduction

1.1 Historical context

Early work on the composition of specifications and programs such as [14, 1] indicated the importance of composition as a key mechanism for the scalability of Formal Methods in software development. Various compositional mechanisms were developed for classical B as defined in [2] and elaborated in [22]. These mechanisms - denoted INCLUDES, EXTENDS, USES, etc. - are syntactic in nature, and concerned with the visibility or inclusion of the text of one machine by another. A variety of visibility and usage rules and constraints are defined. These mechanisms were designed with the scalability of automated proof obligation (PO) generation and proof at least as much in mind as modelling utility. Perhaps unsurprisingly, they are not very intuitive, are dissimilar to inclusion mechanisms in other languages, and not straightforward to use. Later work [23] revealed further unsuspected modelling limitations in the composition of B machines.

Recently completed EU Framework VI project RODIN[1] saw the definition of the Event-B language [19] and the creation of the rich RODIN toolkit [3] for formal modelling, animation, verification, and proof with Event-B. Project RODIN is succeeded by project DEPLOY[2] which will, driven by industrial deployments, further develop the RODIN toolset and Event-B methods.

[1] RODIN - Rigorous Open Development Environment for Open Systems: EU IST Project IST-511599, http://rodin.cs.ncl.ac.uk

[2] DEPLOY - Industrial deployment of system engineering methods providing high dependability and productivity: FP VII Project 214158 under Strategic Objective IST-2007.1.2

The classical B compositional mechanisms have been excised from Event-B to make way for their reinvention in future. The high aspirations of [4], which demonstrated the modelling power that could be unleashed by implementing the rich generic axiomatic structuring of set theory at the metalanguage (as opposed to object language) level, will be realized to some degree by the schedule for project DEPLOY.

The motivation for model decomposition is to reduce model size and proof complexity; there is the bonus of enabling the distribution of development work. Two methods for decompositional working through refinement are on the DEPLOY schedule. Methodologically, they work similarly: a single, "abstract" model M is developed and decomposed - or abstracted - into component models $\{\mathrm{N}_i\}$. The components are refined to more "concrete" versions $\{\mathrm{NR}_i\}$; these concrete refinements are then recomposed into model MR in a *particular way* that guarantees that MR refines M.

[19, 5] propose the state-based decomposition (called "type A" decomposition, after Abrial) of a model: here the state variables $\{v_j\}$ of M are initially partitioned across the $\{\mathrm{N}_i\}$. The events $\{e_k\}$ follow variables they act on into the $\{\mathrm{N}_i\}$. Provided all events acting on a variable v are located in its component machine N_i, that variable is *local*, or *internal* to that machine and needs no special treatment. In general at least one variable w is *shared* between two given component machines that act on it; such a variable is also called *external* to each. If this is not the case, then we simply have disjoint and unrelated developments.

Of course, the refinement of *M* by *MR* only decomposes provided the gluing invariant decomposes conjunctively in the right way. More significantly, [19][3] shows that external variables must be refined by a common, functional gluing invariant; internal variables are not so constrained. The functional constraint is required by the proof of the construction. The part of the gluing invariant concerning say, external v refined by w, can be written $v = h(w)$, and this equality enables certain existential quantifications to be simplified with the existential one-point rule.

The second proposal is for event-based decomposition (called "type B" decomposition, after Butler) from [11, 15]. Since "Event-B machines have the same semantic structure and refinement definitions as action systems" [Op.cit.], this is precisely the reverse of the composition proposal of [10], where it was posed in an action systems [7] setting. Here, an abstract model M is refined in a manner that facilitates the partition of events between component models. The refinement of M to a single model MR decomposes the state variables (by adding new ones), such that MR is expressible as a parallel composition of component models $||\ \{\mathrm{NR}_i\}$ over the partitioned variables. Each event accessing variables in more than one NR_i is decomposed into a set of events each accessing only a local variable, that communicate by message-passing. The semantic correspondence of action systems and CSP is used to proved monotonicity of this process w.r.t. refinement.

Both the above proposals elaborate the traditional "top-down" development process; it remains canonical to start from the most general, and concise abstraction, and then to elaborate through refinement. Such top-down approaches are not natuarally receptive to reuse, where one might want to draw on a database of models and model elements, at

[3] Note that [5] make the stronger requirement that external variables are not data refined, purely to simplify their exposition.

various levels of abstraction and genericity. This work is motivated by the desire to facilitate such working with reuse, i.e. to produce a refinement-preserving compositional method, which reuses existing models. We demonstrate that Event-B models can indeed be so composed, in a manner analogous to the inverse of type A decomposition. Unlike A- and B-decomposition however, *new* events are constructed in the composite machine by a version of the event *fusion* of Butler and Back [15, 8].

This introduction continues with a précis of specification (section 1.2) and refinement (section 1.3) in Event-B, and ends with some remarks (section 1.4) motivating feature-based composition as a form of reuse. Section 2 then defines our form of model composition, including the mechanism of event fusion. We show that model consistency is preserved under such composition. Section 3 proves that fusion preserves refinement, an essential property for scalable working. In conclusion section 4 considers related work, and describes future work.

1.2 Event-B Basics

This section is a précis of parts of [19], the Event-B language definition.

Event-B is designed for long-running *reactive* hardware/software systems that respond to stimuli from user and/or environment. The set-theoretic language in first-order logic (FOL) takes as semantic model a transition system labelled with event names. The correctness of a model is defined by an invariant property, i.e. a predicate, or constraint, which every state in the system must satisfy. More practically, every event in the system must be shown to preserve this invariant; this verification requirement is expressed in a number of *proof obligations* (POs). In practice this verification is performed either by model checking or theorem proving (or both).

For modelling in Event-B the two units of structuring are the *machine* of dynamic variables, events and their invariants, and the *context* of static data of sets, constants and their axioms. Every machine *sees* at least one context. The unit of behaviour is the *event*. An event e acting on (a list of) state variables v, subject to enabling condition, or *guard* $G(v)$ and *action*, or *assignment* $E(v)$, has syntax

$$e \mathrel{\widehat{=}} \textsf{when } G(v) \textsf{ then } E(v) \textsf{ end} \tag{1}$$

That is, when the state is such that the guard is true, this enables the action, or state transition defined by $E(v)$. Next we give a more general syntax for a nondeterministic event. We give the guard, whose meaning is obvious from the before-after predicate for the event: the guard is precisely the statement that there exists an after-state defined by the before-after predicate, i.e. that the latter is *feasible*.

$$\text{event syntax:} \qquad \textsf{any } t \textsf{ where } Q(t,v) \textsf{ then } v := F(t,v) \textsf{ end} \tag{2}$$

$$\text{guard:} \qquad \exists\, t \bullet Q(t,v) \tag{3}$$

$$\text{before-after predicate:} \qquad \exists\, t \bullet (Q(t,v) \wedge v' = F(t,v)) \tag{4}$$

Note the shorthand syntax: since v above is in general a variable list, $F(t,v)$ is an expression list. (2-4) define a t-indexed nondeterministic choice between those transitions

$v' = F(t, v)$ for which $Q(t, v)$ is true[4]. t is interpreted as an input from the environment. Syntactic sugar is available: parallel ($||$) is used to enumerate multiple single-variable assignments. In the any form, the event guard is not stated explicitly since it is constructed automatically from the where clause $Q(t, v)$. The following useful property always holds for the guard G_e and before-after predicate E_e of an any -defined event e:

$$E_e \Rightarrow G_e \tag{5}$$

For the sake of completeness it is worth defining a more general event syntax that specifies an after-state in terms of a predicate it satisfies, called $x :| P(x, x', y)$. The equality-based event definition of (2-4) is usually sufficiently expressive and forms the basis of this work.

event syntax:	any t where $P(x, t, y)$ then $x := t$ end	(6)
guard:	$\exists x' \bullet P(x, x', y)$	(7)
before-after predicate:	$P(x, x', y)$	(8)

An event e works in a model (comprising a machine and at least one context) with constants c and sets s subject to *axioms* (properties) $P(s, c)$ and an *invariant* $I(s, c, v)$. Thus the event guard G and assignment with before-after predicate E take s, c as parameters. Two of the consistency proof obligations [5] (POs) for event e are FIS (feasibility preservation) and INV (invariant preservation). For an event defined as (2-4), FIS clearly discharges trivially.

$$P(s, c) \wedge I(s, c, v) \wedge G(s, c, v) \Rightarrow \exists v' \bullet E(s, c, v, v') \qquad \text{FIS} \tag{9}$$

$$P(s, c) \wedge I(s, c, v) \wedge G(s, c, v) \wedge E(s, c, v, v') \Rightarrow I(s, c, v') \qquad \text{INV} \tag{10}$$

1.3 Refinement

The refinement of a context is simply its elaboration, by the addition of new sets, constants and axioms. The refinement of a machine includes both data and algorithm refinement: all variables v are replaced by new ones w, some simply by renaming - i.e. of the same type and meaning - and others by variables of different type. Existing events are transformed to work on the new variables, and new events can be defined; that is, the behaviour of an abstract event e can be refined by some sequence of e and new events. The new behaviour will usually reduce nondeterminism. When model $N(w)$ refines $M(v)$, it also has an invariant $J(s, c, v, w)$ which can include M's variables v. This "gluing invariant", or refinement relation, has the function of relating abstract variables v to concrete ones w mathematically.

In Fig. 1, M sees C, N refines M and D refines C, then N sees D. It is also possible for C not to be refined (i.e. to be identity-refined), in which case N sees C.

As for simple machines, there are proof obligations for refinement. We assume axioms $P(s, c)$, and abstract, concrete invariants $I(s, c, v)$ and $J(s, c, v, w)$ respectively. An

4 The deterministic assignment is simply written $v := F(v)$, without an any variable or where clause.

5 See [19] for the others.

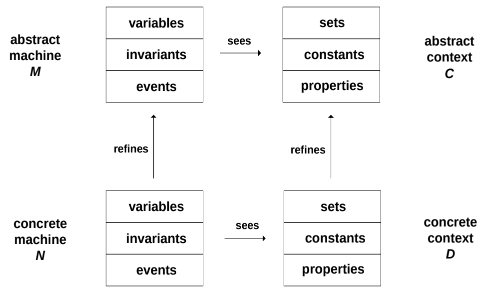

Fig. 1. Machine and context refinements (from [19])

abstract event with guard $G_A(s,c,v)$ and before-after predicate $E_A(s,c,v,v')$ is refined by a concrete event with guard $G_C(s,c,w)$ and before-after predicate $E_C(s,c,w,w')$. The following obligations state that the concrete event is feasible (FIS_REF), the concrete guard strengthens the abstract one (GRD_REF), and that every concrete step is correct (simulates) w.r.t. some abstract step (INV_REF):

$$P(s,c) \land I(s,c,v) \land J(s,c,v,w) \land G_C(s,c,w)$$
$$\Rightarrow \exists\, w' \bullet E_C(s,c,w,w') \qquad \text{FIS\_REF} \qquad (11)$$
$$P(s,c) \land I(s,c,v) \land J(s,c,v,w) \land G_C(s,c,w)$$
$$\Rightarrow G_A(s,c,v) \qquad \text{GRD\_REF} \qquad (12)$$
$$P(s,c) \land I(s,c,v) \land J(s,c,v,w)$$
$$\land\, G_C(s,c,w) \land E_C(s,c,w,w')$$
$$\Rightarrow \exists\, v' \bullet (E_A(s,c,v,v') \land J(s,c,v',w')) \qquad \text{INV\_REF} \qquad (13)$$

[19] defines further refinement obligations, for nondivergence of new events introduced, and “relative deadlockfreeness” to ensure a concrete model cannot deadlock more often than the abstract one. We do not pursue these matters in this work.

1.4 Model reuse with features

A useful way to analyse reuse - provided by the Software Product Line (SPL) community, e.g. [20] - is in terms of data and behavioural variability [12] between system versions. The concepts are as applicable in software reuse through evolution as they are in SPLs. Event-B deals with static data variability by separating - in a B model - the dynamic *machine* from the static *context*. However, there is no mechanism to deal with behavioural variability. It is straightforward to generate variant versions of a B development that differ only in configuration, or static data: simply switch the required contexts into the refinement tree. In [21] we proposed the notion of a feature model, as a fine level of granularity for B specification, with composition of such features as a mechanism for behavioural variability in development, thus contributing to the “Roadmap for Enhanced Languages and Methods to Aid Verification” [18]. A *feature* is defined simply to be a B model which is (largely) atomic with respect to composition. “Atomic”, in the sense that no syntactically partial, or incomplete, model will (at this time) be input to reuse in modelling. “Largely”, in the sense that certain obvious refactorings, such as systematic renamings of certain identifiers, or text insertions such as strengthening of predicates, will be allowed.

By way of brief motivation for feature-based composition, imagine a database of Event-B features for some application domain, such as resource management for distributed computing. Imagine a model M_1 with variables x, y and an event

$$e = \textsf{ any } t \textsf{ where } Q_1(t,x) \textsf{ then } x := F_1(t,x,y) \textsf{ end}$$

where the event specifies the allocation of some resource x such as a virtual circuit, subject to some QoS requirement given by (Q_1, F_1). We might wish - subject to suitable systematic variable and event renaming - to compose M_1 with some other model M_2 to allow specification of other resources through other events f, g in M_2. If both variable lists and event lists are disjoint, composition is a trivial matter with no extra proof obligations arising. Should the variable lists overlap, it will be necessary to show that M_1 events preserve the invariant of M_2 and vice versa.

A more interesting case is where we wish to *fuse* an event e from M_1 with some event f from M_3, say. It may be that M_3 specifies different requirements of virtual circuits, such as fault-tolerance/redundancy. We may wish to select in x a virtual circuit satisfying QoS (Q_1, F_1), *and* supporting fault-tolerance as specified by f.

In the next section we demonstrate that Event-B models (or features) can indeed be composed, in a manner analogous to the inverse of state-based decomposition, in a way that preserves refinement. We will focus in particular on the case of event fusion.

2 Model and event fusion

Consider two models M_1 and M_2 which we propose to *fuse* by combining variables and events. That is, we concatenate the variable lists and events, conjoin those events with common names (in a manner to be defined) in a new model M. The variable list v in M_1 comprises the list x of actioned variables and the list y of skipping variables for each event[6]. Similarly variables w in M_2 comprise actioned z and skipping a. We define $xz = x \cap z$, the common actioned variables, and $ya = y \cap a$, the common skipping variables. Note that the other intersecting variable lists yz and xa are both empty, to enable meaningful composition definitions. Since the context axioms P_1, P_2 of the two models do not influence the proofs we assume they share sets and constants s, c without loss of generality.

[6] Strictly speaking v should be partitioned into (x_e, y_e) for each event e. We do not need this decoration since only one event in each model is considered.

$M_1 : v = x \cup y$
    $s, c, P_1(s, c)$ context
    $v, I_1(s, c, v)$ invariant
event:
    $e =$ any α where $Q_1(\alpha, v)$
        then $x := F_1(\alpha, v)$
Thus
    $G_e \mathrel{\widehat{=}} \exists \alpha \bullet Q_1(\alpha, v)$
    $E_e \mathrel{\widehat{=}} \exists \alpha \bullet (Q_1(\alpha, v) \wedge x' = F_1(\alpha, v)) \wedge y' = y$

$M_2 : w = z \cup a$
    $s, c, P_2(s, c)$ context
    $w, I_2(s, c, w)$ invariant
event:
    $f =$ any β where $Q_2(\beta, w)$
        then $z := F_2(\beta, w)$
Thus
    $G_f \mathrel{\widehat{=}} \exists \beta \bullet Q_2(\beta, w)$
    $E_f \mathrel{\widehat{=}} \exists \beta \bullet (Q_2(\beta, w) \wedge z' = F_2(\beta, w)) \wedge a' = a$

Next we define the fused model M, distinguishing clearly in the before-after predicate between actioned variables $< x - xz >$ exclusive to M_1, common actioned variables xz, and actioned variables $< z - xz >$ exclusive to M_2. We write the fusion of events e and f as $e \odot f$. The fused model is then specified in the obvious way:

$M : v, w = x \cup z \cup y \cup a$
$s, c, P_1(s, c) \wedge P_2(s, c)$ context
$v, w, I_1(s, c, v) \wedge I_2(s, c, w)$ invariant
$e \odot f =$ any α, β where $Q_1(\alpha, v) \wedge Q_2(\beta, w)$
        then $x := F_1(\alpha, v) \mid\mid z := F_2(\beta, w)$
        end

The usual existence proof obligation for a machine context - i.e. $P_1 \wedge P_2$ - arises here.

The meaning of the above syntax - i.e. the use of $\mid\mid$ over intersecting variable lists, undefined as yet in the Event-B language - is given by the fused guard and before-after predicate definitions[7]:

$$G_{e \odot f} \mathrel{\widehat{=}} \exists \alpha, \beta \bullet (Q_1(\alpha, v) \wedge Q_2(\beta, w) \wedge F_1(\alpha, v) = F_2(\beta, w)) \tag{14}$$

$$\begin{aligned} E_{e \odot f} \mathrel{\widehat{=}} \exists \alpha, \beta \bullet (&Q_1(\alpha, v) \wedge Q_2(\beta, w) \wedge < x - xz >' = F_1(\alpha, v) \wedge \\ &xz' = F_1(\alpha, v) \wedge xz' = F_2(\beta, w) \wedge < z - xz >' = F_2(\beta, w)) \wedge \\ &y' = y \wedge a' = a \end{aligned} \tag{15}$$

Clearly, there must be sufficient nondeterminism in these definitions to satisfy $G_{e \odot f}$ for meaningful state values v, w.

The following useful properties are obvious:

$$G_{e \odot f} \Rightarrow G_e \wedge G_f \qquad E_{e \odot f} \Rightarrow E_e \wedge E_f \tag{16}$$

Theorem 1 Event consistency (9-10) is preserved under model fusion.
Proof Assume $P_1 \wedge P_2 \wedge I_1 \wedge I_2 \wedge G_{e \odot f} \wedge E_{e \odot f}$. From (16) the hypotheses of INV(e) and INV(f) are made available, and it follows that $I_1(s, c, v') \wedge I_2(s, c, w')$. **QED**

We make some observations:

[7] $G_{e \odot f}, E_{e \odot f}$ definitions are given in shorthand; F_1, F_2 are expression *lists*, each list being partitioned according to the variable sublists in use at that point in the definitions. Thus (14-15) should be read in terms of the appropriate sublists. In particular, (14) refers only to the the sublists of F_1, F_2 assigning to common actioned variables xz.

1. The fusion of two models clearly requires sufficient nondeterminism in the fusing events' actions over shared variables, in order for the fused event to be feasible (and the fused guard not vacuously false). A natural way in which this might arise is as follows. Event $e(v_1, v_2, v_3)$, say, assigns v_1 nondeterministically to $F_1(\alpha, v_1, v_2, v_3)$ for some α, v_2 to anything in its type V_2, and skips on v_3. Event $f(v_1, v_2, v_3)$ assigns v_1 to anything in its type V_1, v_2 nondeterministically to $F_2(\beta, v_1, v_2, v_3)$ for some β, and skips on v_3. This represents the compositional modelling, from prior component models, of the requirement to perform F_1 on v_1 and F_2 on v_2, in the manner suggested in section 1.4.
   Methodologically it is desirable that the fusion of two events should refine each of them, and this is indeed the case, as we show below.
2. **Theorem 1a** Theorem 1 applies for two models with disjoint variable lists and composing events by the same reasoning. This is a parallel composition of models, where each composed event represents the product of all transitions on all variables from the component models
3. **Theorem 1b** Theorem 1 applies for two models with disjoint variable and event lists; this is the embedding of each model in a larger one, where each event acts on variables from its own model and skips on those from the other model. The POs discharge trivially since each event is the identity refinement of its abstract counterpart: for event e acting on v in composed model M, INV discharges by noting that $I_1(s, c, v')$ follows from INV(e) in M_1, and that $I_2(s, c, w)$ follows from skip in M_2.

Theorem 2 The fusion $e \odot f$, in model M, of two events e and f refines each of those events in their respective models.
Proof We discharge the refinement obligations (11-13) for $e \sqsubseteq e \odot f$; the f case is treated identically. FIS_REF (11) follows trivially in the same way that FIS does for events of the form (2-4), since $G_{e\odot f} \Rightarrow \exists\, v', w' \bullet E_{e\odot f}$. GRD_REF (12) follows trivially since $G_{e\odot f} \Rightarrow G_e$. For INV_REF, for clarity we rename abstract variables in M_v v_0, and assume

$$P_1 \wedge P_2 \wedge I_1(v_0) \wedge v_0 = v \wedge I_1(v) \wedge I_2(w) \wedge G_{e\odot f}(v, w) \wedge E_{e\odot f}(v, w, v', w')$$

We must prove

$$\exists\, v'_0 \bullet (E_e(v_0, v'_0) \wedge v' = v'_0 \wedge I_1(v'_0) \wedge I_2(w'))$$

that is, removing the identical-copy abstract variables v_0, v'_0

$$E_e(v, v') \wedge I_1(v') \wedge I_2(w'))$$

Since $E_{e\odot f} \Rightarrow E_e$, and we have the second two conjuncts from INV(e) and INV(f) resp. we are done. **QED**

3 Preservation of refinement by event fusion

We show that the fusion of refined events refines the fusion of the original events. Consider the compositional arrangement of models in Fig. 2. Since this construction is inspired by the state-based decomposition construction of [19] (as discussed in section 1.1), the diligent reader will see that the gluing invariants here are precisely those of [Op.cit.].

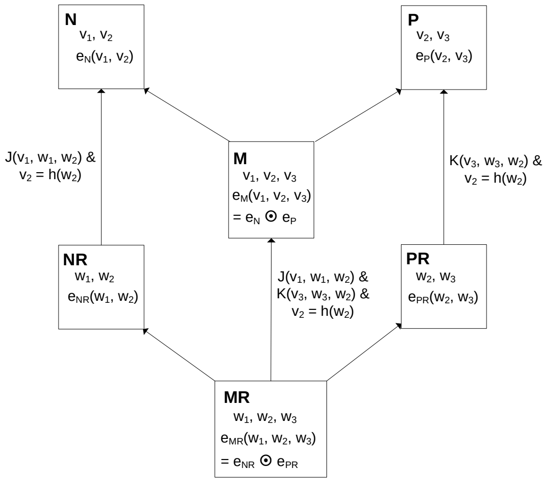

Fig. 2. Refinement of event fusion

Model N has variables v_1, v_2 and event $e_N(v_1, v_2)$ with guard G_N and before-after predicate E_N. Model P has variables v_2, v_3 and event $e_P(v_2, v_3)$ with guard G_P and before-after predicate E_P. v_2 is thus the shared variable between N and P. Model M over variables v_1, v_2, v_3 with event $e_M(v_1, v_2, v_3) \widehat{=} e_N \odot e_P$ is the fusion[8] of N and P. The guard and before-after predicate of e_M are named G_M, E_M respectively.

Next we have two models NR, PR which refine N, P respectively. NR has variables w_1, w_2 and event $e_{NR}(w_1, w_2)$ with guard G_{NR} and before-after predicate E_{NR}. e_{NR} refines e_N with gluing invariant

$$J(v_1, w_1, w_2) \wedge v_2 = h(w2) \tag{17}$$

Similarly, PR has variables w_2, w_3 and event $e_{PR}(w_2, w_3)$ with guard G_{PR} and before-after predicate E_{PR}. e_{PR} refines e_P with gluing invariant

$$K(v_3, w_3, w_2) \wedge v_2 = h(w2) \tag{18}$$

Note the requirement that the shared variable is refined in the same functional manner in both machines; this satisfies the intuition that a shared variable should be treated "in the same way" in each sharing refinement chain, before the refinements are fused. The local variables in component machines may be defined more generally and independently of each other, while allowing the reference to the concrete shared variable.

[8] Contrast this construction with that of [19] as outlined in sec. 1.1: in that case events e_N and e_P, both acting on external variable v_2, *both* appear independently in M. Then, in N the external effect of e_P on v_2 must be modelled by a new *external* event e_{Px}; in N e_{Px} abstracts e_P in M. The new event is required in order that M refines N. In our scheme the fusion of events removes their independence of behaviour, thus removing the need for external events. The proof of the construction is simplified, but still requires the functional gluing invariant, for the same reason as [Op.cit.].

Finally, model MR over variables w_1, w_2, w_3 has event $e_{MR}(w_1, w_2, w_3) \mathrel{\widehat{=}} e_{NR} \odot e_{PR}$ which is the fusion of e_{NR} and e_{PR}. We say that e_{MR} has guard G_{MR} and before-after predicate E_{MR}. We must now show that MR refines M w.r.t. gluing invariant

$$J(v_1, w_1, w_2) \wedge v_2 = h(w2) \wedge K(v_3, w_3, w_2) \tag{19}$$

Theorem 3 Given that $\underline{e_N \sqsubseteq e_{NR}}$:

$$\begin{aligned}&J(v_1, w_1, w_2) \wedge v_2 = h(w_2) \wedge G_{NR}(w_1, w_2) \wedge E_{NR}(w_1, w_2, w_1', w_2') \\ &\Rightarrow G_N(v_1, v_2) \wedge \exists\, v_1', v_2' \bullet (E_N(v_1, v_2, v_1', v_2') \wedge J(v_1', w_1', w_2') \wedge v_2' = h(w_2')\end{aligned} \tag{20}$$

and $\underline{e_P \sqsubseteq e_{PR}}$:

$$\begin{aligned}&K(v_3, w_3, w_2) \wedge v_2 = h(w_2) \wedge G_{PR}(w_2, w_3) \wedge E_{PR}(w_2, w_3, w_2', w_3') \\ &\Rightarrow G_P(v_2, v_3) \wedge \exists\, v_2', v_3' \bullet (E_P(v_2, v_3, v_2', v_3') \wedge K(v_3', w_3', w_2') \wedge v_2' = h(w_2')\end{aligned} \tag{21}$$

we must show[9] $\underline{e_M = e_N \odot e_P \sqsubseteq e_{MR} = e_{NR} \odot e_{PR}}$:

$$\begin{aligned}&J(v_1, w_1, w_2) \wedge K(v_3, w_3, w_2) \wedge v_2 = h(w_2) \wedge \\ &\quad G_{MR}(w_1, w_2, w_3) \wedge E_{MR}(w_1, w_2, w_3, w_1', w_2', w_3') \\ &\Rightarrow G_M(v_1, v_2, v_3) \wedge \exists\, v_1', v_2', v_3' \bullet (E_M(v_1, v_2, v_3, v_1', v_2', v_3') \wedge \\ &\quad J(v_1', w_1', w_2') \wedge K(v_3', w_3', w_2') \wedge v_2' = h(w_2'))\end{aligned} \tag{22}$$

Proof is straightforward and uses the fusion definitions (14, 15), i.e.

$$G_N \mathrel{\widehat{=}} \exists\, \alpha \bullet Q_N(\alpha, v_1, v_2) \tag{23}$$

$$E_N \mathrel{\widehat{=}} \exists\, \alpha \bullet (Q_N \wedge v_1' = F_N^1(\alpha, v_1, v_2) \wedge v_2' = F_N^2(\alpha, v_1, v_2)) \tag{24}$$

$$G_P \mathrel{\widehat{=}} \exists\, \beta \bullet Q_P(\beta, v_2, v_3) \tag{25}$$

$$E_P \mathrel{\widehat{=}} \exists\, \beta \bullet (Q_P \wedge v_2' = F_P^2(\beta, v_2, v_3) \wedge v_3' = F_P^3(\beta, v_2, v_3)) \tag{26}$$

$$G_{NR} \mathrel{\widehat{=}} \exists\, \gamma \bullet Q_{NR}(\gamma, w_1, w_2) \tag{27}$$

$$E_{NR} \mathrel{\widehat{=}} \exists\, \gamma \bullet (Q_{NR} \wedge w_1' = F_{NR}^1(\gamma, w_1, w_2) \wedge w_2' = F_{NR}^2(\gamma, w_1, w_2)) \tag{28}$$

$$G_{PR} \mathrel{\widehat{=}} \exists\, \delta \bullet Q_{PR}(\delta, w_2, w_3) \tag{29}$$

$$E_{PR} \mathrel{\widehat{=}} \exists\, \delta \bullet (Q_{PR} \wedge w_2' = F_{PR}^2(\delta, w_2, w_3) \wedge w_3' = F_{PR}^3(\delta, w_2, w_3)) \tag{30}$$

and thus

$$G_M \mathrel{\widehat{=}} \exists\, \alpha, \beta \bullet (Q_N(\alpha, v_1, v_2) \wedge Q_P(\beta, v_2, v_3) \wedge F_N^2 = F_P^2) \tag{31}$$

$$E_M \mathrel{\widehat{=}} \exists\, \alpha, \beta \bullet (Q_N \wedge Q_P \wedge v_1' = F_N^1 \wedge v_2' = F_N^2 \wedge v_2' = F_P^2 \wedge v_3' = F_P^3) \tag{32}$$

$$G_{MR} \mathrel{\widehat{=}} \exists\, \gamma, \delta \bullet (Q_{NR} \wedge Q_{PR} \wedge F_{NR}^2 = F_{PR}^2) \tag{33}$$

$$\begin{aligned}E_{MR} \mathrel{\widehat{=}} \exists\, \gamma, \delta \bullet (&Q_{NR} \wedge Q_{PR} \wedge w_1' = F_{NR}^1 \wedge w_2' = F_{NR}^2 \wedge w_2' = F_{PR}^2 \wedge \\ &w_3' = F_{PR}^3)\end{aligned} \tag{34}$$

[9] [19] states that, instead of discharging (11-13), it suffices to prove the composite statement (22).

We rewrite the theorem in expanded form, omitting redundant guard expressions by (5), i.e.

Given that $\underline{e_N \sqsubseteq e_{NR}}$:

$$\begin{aligned}
&J(v_1, w_1, w_2) \wedge v_2 = h(w_2) \wedge \\
&\exists \gamma \bullet (Q_{NR}(\gamma, w_1, w_2) \wedge w_1' = F_{NR}^1(\gamma, w_1, w_2) \wedge w_2' = F_{NR}^2(\gamma, w_1, w_2)) \\
&\Rightarrow \exists v_1', v_2' \bullet (\exists \alpha \bullet (Q_N(\alpha, v_1, v_2) \wedge v_1' = F_N^1(\alpha, v_1, v_2) \wedge \\
&\qquad v_2' = F_N^2(\alpha, v_1, v_2)) \wedge J(v_1', w_1', w_2') \wedge v_2' = h(w_2'))
\end{aligned} \tag{35}$$

... where the RHS can be simplified to ...

$$\exists \alpha \bullet (Q_N(\alpha, v_1, v_2) \wedge J(F_N^1(\alpha, v_1, v_2), w_1', w_2') \wedge F_N^2(\alpha, v_1, v_2) = h(w_2')) \tag{36}$$

and $\underline{e_P \sqsubseteq e_{PR}}$:

$$\begin{aligned}
&K(v_3, w_3, w_2) \wedge v_2 = h(w_2) \wedge \\
&\exists \delta \bullet (Q_{PR}(\delta, w_2, w_3) \wedge w_2' = F_{PR}^2(\delta, w_2, w_3) \wedge w_3' = F_{PR}^3(\delta, w_2, w_3)) \\
&\Rightarrow \exists v_2', v_3' \bullet (\exists \beta \bullet (Q_P(\beta, v_2, v_3) \wedge v_2' = F_P^2(\beta, v_2, v_3) \wedge \\
&\qquad v_3' = F_P^3(\beta, v_2, v_3)) \wedge K(v_3', w_3', w_2') \wedge v_2' = h(w_2'))
\end{aligned} \tag{37}$$

... where the RHS can be simplified to ...

$$\exists \beta \bullet (Q_P(\beta, v_2, v_3) \wedge K(F_P^3(\beta, v_2, v_3), w_3', w_2') \wedge F_P^2(\beta, v_2, v_3) = h(w_2')) \tag{38}$$

we must show $\underline{e_M \sqsubseteq e_{MR}}$:

$$\begin{aligned}
&J(v_1, w_1, w_2) \wedge K(v_3, w_3, w_2) \wedge v_2 = h(w_2) \wedge \\
&\exists \gamma, \delta \bullet (Q_{NR}(\gamma, w_1, w_2) \wedge Q_{PR}(\delta, w_2, w_3) \wedge w_1' = F_{NR}^1(\gamma, w_1, w_2) \wedge \\
&\qquad w_2' = F_{NR}^2(\gamma, w_1, w_2) \wedge w_2' = F_{PR}^2(\delta, w_2, w_3) \wedge w_3' = F_{PR}^3(\delta, w_2, w_3)) \\
&\Rightarrow \exists v_1', v_2', v_3' \bullet (\exists \alpha, \beta \bullet (Q_N(\alpha, v_1, v_2) \wedge Q_P(\beta, v_2, v_3) \wedge v_1' = F_N^1(\alpha, v_1, v_2) \wedge \\
&\qquad v_2' = F_N^2(\alpha, v_1, v_2) \wedge v_2' = F_P^2(\beta, v_2, v_3) \wedge \\
&\qquad v_3' = F_P^3(\beta, v_2, v_3)) \wedge \\
&\qquad J(v_1', w_1', w_2') \wedge K(v_3', w_3', w_2') \wedge v_2' = h(w_2'))
\end{aligned} \tag{39}$$

... where the RHS can be simplified to ...

$$\begin{aligned}
&\exists \alpha, \beta \bullet (Q_N(\alpha, v_1, v_2) \wedge Q_P(\beta, v_2, v_3) \wedge \\
&\qquad F_N^2(\alpha, v_1, v_2) = F_P^2(\beta, v_2, v_3) \wedge J(F_N^1(\alpha, v_1, v_2), w_1', w_2') \wedge \\
&\qquad K(F_P^3(\beta, v_2, v_3), w_3', w_2') \wedge F_P^2(\beta, v_2, v_3) = h(w_2'))
\end{aligned} \tag{40}$$

Assuming the hypothesis (39) for the refinement of e_M, we can partition its terms into separate quantifications over γ and δ, and thus infer the hypotheses (35, 37) for the refinements of e_N, e_P respectively. The consequents of the component refinements (36, 38) follow, and then we infer the result (40) directly by recombining the terms under a joint quantification over α, β. **QED**

4 Conclusion and Related Work

(De-)Compositional approaches to modelling and verification have been extensively studied for obvious reasons, and continue to be developed. We discuss only those most relevant to Event-B; whilst contemporary work on component- and service-based composition such as [6] is interesting, its application to Event-B remains for the future.

Following earlier work on temporal property verification on labelled transition systems (LTS) inspired by B [9, 13], Kouchnarenko and Lanoix [16, 17] investigated compositional verification in that LTS setting. For their expressive "constraint synchonized product" composition of components, preservation of both local and global invariants is shown, as well as compositionality of refinement. With stuttering behaviour allowed and non-increasing of deadlocks in refinement, their work is of interest to the Event-B community. While this work does not deal with these behavioural aspects of refinement - leaving that for the future - it does allow for intersecting state spaces, i.e. communication through shared variables. Kouchnarenko and Lanoix have disjoint state spaces but their work may be extensible to a message-passing composition like that of Butler [11].

Patterns and techniques for compositional/decompositional working with Event-B are in their infancy, reflecting the fact that they remain to be implemented in what is still a very recent language and method. Although the decompositional techniques of section 1 have been known for some years, and paper-based case studies have been published, e.g. [11], these techniques remain to be implemented by tools. Some progress is expected in this regard during project DEPLOY. For these more established techniques, and certainly for newer proposals such as ours, case study work is required to validate their utility, followed by prototype tool development to implement them.

In the short term we will investigate the extensibility of the results of this work. Obvious questions are (i) does the construction work for full behavioural Event-B refinement (as mentioned in sec. 1.3), (ii) under what conditions can features expressed in the more general syntax (6-8) be composed, and (iii) can we compose subject to less constrained gluing invariants than (17-18)? Beyond that we anticipate the proposal of more elaborate patterns of composition. Our proposal (per Fig. 2) gives a simple one-to-one feature refinement pattern, in general inadequate for elaborating an architectural model of the system. More flexibility is required in the elaboration of the modular arrangement of refinements. In the figure, for example, we can imagine different depths in the feature refinement chains, or feature decompositions: say that PR is refined by event-based decomposition into PR_{21}, PR_{22}, and each of these is further refined into PR_{31}, PR_{32} before fusing, together with NR, into MR.

Significant further tool infrastructure will ultimately be required to support reuse, i.e. the construction of system variants from different arrangements of feature composition and refinement. This includes inter alia system variant identification (in terms of components), feature refactoring, and proliferation of feature changes.

References

[1] M. Abadi and L. Lamport. Composing specifications. *ACM Trans. Program. Lang. Syst.*, 15(1):73–132, 1993.

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

[3] J. R. Abrial, M. Butler, S. Hallerstede, and L. Voisin. An open extensible tool environment for Event-B. In Z. Liu and J. He, editors, *Proc. ICFEM 2006*, volume 4260 of *LNCS*, Macau, 2006.

[4] J.-R. Abrial, D. Cansell, and G. Laffitte. "Higher-order" mathematics in B. In D. Bert, J.P. Bowen, M.C. Henson, and K. Robinson, editors, *Second International Conference of B and Z Users*, volume 2272 of *LNCS*, page 370393, Grenoble, France, January 2002. Springer.

[5] J.-R. Abrial and S. Hallerstede. Refinement, decomposition and instantiation of discrete models: Application to Event-B. *Fundamenta Informaticae*, 77(1-2), 2007.

[6] C. Attiogbé, P. André, and G. Ardourel. Checking component composability. In *SC06:5th International Symposium on Software Composition*, volume 4089 of *LNCS*. Springer, 2006.

[7] R.-J. Back and R. Kurki-Suonio. Decentralization of process nets with centralized control. *Distributed Computing*, 3(2):73–87, 1989.

[8] R.J.R. Back and M. Butler. Fusion and simultaneous execution in the refinement calculus. *Acta Informatica*, 35:921–949, 1998.

[9] F. Bellegarde, J. Julliand, and O. Kouchnarenko. Ready-simulation is not ready to express a modular refinement relation. In *Fundamental Aspects of Software Engineering 2000, FASE'2000*, volume 1783 of *LNCS*, pages 266–283, Berlin, March 2000.

[10] M. Butler. Stepwise refinement of communicating systems. *Science of Computer Programming*, 27:139–173, 1996.

[11] M. Butler. An approach to the design of distributed systems with B AMN. In J.P. Bowen, M.G. Hinchey, and D. Till, editors, *10th International Conference of Z Users*, volume 1212 of *LNCS*, pages 223–241, Reading, UK, 1997. Springer.

[12] J. Coplien, D. Hoffman, and D. Weiss. Commonality and variability in software engineering. *IEEE Software*, pages 37–45, November/December 1998.

[13] C. Darlot, J. Julliand, and O. Kouchnarenko. Refinement preserves PLTL properties. In *Third International Conference of B and Z Users ZB'03 - Formal Specification and Development in Z and B*, volume 2651 of *LNCS*, pages 408–420, Turku, Finland, June 2003.

[14] C.B. Jones. Tentative steps toward a development method for interfering programs. *ACM Transactions on Programming Languages and Systems*, 5(4):596–619, October 1983.

[15] C.B.(ed.) Jones. Intermediate report on methodology. Technical Report Deliverable 19, EU Project IST-511599 - RODIN, August 2006. http://rodin.cs.ncl.ac.uk.

[16] O. Kouchnarenko and A. Lanoix. Refinement and verification of synchronized component-based systems. In K. Araki, S. Gnesi, and D. Mandrioli, editors, *Proc. FME2003: Formal Methods*, volume 2805 of *LNCS*, pages 341–358, Pisa, Italy, September 2003. Springer.

[17] O. Kouchnarenko and A. Lanoix. Verifying invariants of component-based systems through refinement. In C. Rattray, S. Maharaj, and C. Shankland, editors, *AMAST'04:10th Int. Conf. on Algebraic Methodology and Software Technology*, volume 3116 of *LNCS*, pages 289–303, Stirling, Scotland, July 2004. Springer.

[18] Abrial J. R. Batory D. Butler M. Coglio A. Fisler K. Hehner E. Jones C. B. Miller D. Peyton-Jones S. Sitaraman M. Smith D. R. Leavens, G. T. and A. Stump. Roadmap for enhanced languages and methods to aid verification. In *Proc. 5th Int. Conf. Generative Programming and Component Engineering*, Portland, Oregon, 2006.

[19] C. Métayer, J.-R. Abrial, and L. Voisin. Event-B Language. Technical Report Deliverable 3.2, EU Project IST-511599 - RODIN, May 2005. http://rodin.cs.ncl.ac.uk.

[20] K. Pohl, G. Boeckle, and F. van der Linden. *Software Product Line Engineering Foundations, Principles, and Techniques*. Springer, Heidelberg, 2005.

[21] M. Poppleton. Towards feature-oriented specification and development with Event-B. In P. Sawyer, B. Paech, and P. Heymans, editors, *Proc. REFSQ 2007: Requirements Engineering: Foundation for Software Quality*, volume 4542 of *LNCS*, pages 367–381, Trondheim, Norway, June 2007. Springer.

[22] M.-L. Potet. Spécifications et développements structurés dans la méthode B. *Technique et Science Informatiques*, 22:61–88, Février 2003.

[23] M.-L. Potet and Y. Rouzaud. Composition and refinement in the B-method. In D. Bert, editor, *2nd International B Conference*, volume 1393 of *LNCS*, pages 46–65, Montpellier, France, April 1998. Springer.