Refinement of Decomposed Models by Interface Instantiation

Stefan Hallerstede[a,*], Thai Son Hoang[b]

[a]*Aarhus University, Denmark*
[b]*ETH Zurich, Switzerland*

Abstract

Decomposition is a technique to separate the design of a complex system into smaller sub-models, which improves scalability and team development. In the shared-variable decomposition approach for Event-B, sub-models share external variables and communicate through external events which cannot be easily refined.

Our first contribution hence is a proposal for a new construct called interface that encapsulates the external variables, along with a mechanism for interface instantiation. Using the new construct and mechanism, external variables can be refined consistently. Our second contribution is an approach for verifying the correctness of Event-B extensions using the supporting Rodin tool. We illustrate our approach by proving the correctness of interface instantiation.

Keywords: Event-B, Decomposition, Refinement, External variables, Interface instantiation

1. Introduction

When decomposing a model into sub-models we intend to continue refining the sub-models independently of each other while preserving the properties of the full model. A suitable decomposition method for Event-B has been proposed by Abrial [1]. It partitions events of a model between its sub-models. Variables of the model are split correspondingly into external variables shared by the sub-models and internal variables private to each sub-model. For all external variables of a sub-model, external events that mimic the effect of corresponding (internal) events of other sub-models have to be added. If we want to refine external variables, we have to provide a gluing invariant that is functional, say, $v = h(w)$ where v are the abstract variables and w the concrete variables. Abrial [1] also proposes to rewrite the external events with $v := h(w)$ so that concrete and abstract events are equivalent. Internal variables and internal events are refined as usual in Event-B [2].

[*]Corresponding author

We propose a practical method for external event refinement that aids in structuring and understanding complex models. This requires a trade-off between generality and practicality. We believe that it would be difficult to generalise the method that we propose without sacrificing its practicality. The theory of the method is not difficult. Our aim is to make using a difficult technique, the refinement of external variables and external events in Event-B, as easy as possible.

We call a collection of external variables with the external invariants an *interface*. Modelling interfaces "manually" by marking the corresponding variables as being external and refining them by specifying functional invariants makes it difficult to decompose and refine a model repeatedly. Fig. 1 illustrates the problem where a model M is decomposed three times and the resulting sub-models are refined. We are interested in the two sub-models $\mathsf{M_1}$ and $\mathsf{M_2}$ at the bottom. How do we find the shared external invariants?

The lists of variables w_1, w_2, v_1 and v_2 are not necessarily disjoint. Let w be

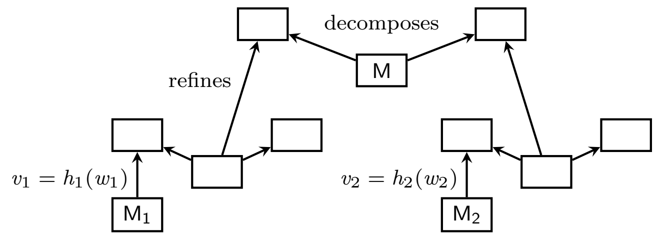

Figure 1: Maintaining the external invariants of several sub-models "manually"

the list of variables occurring in w_1 or w_2 and v be the list of variables occurring in v_1 or v_2. We need to find one suitable external invariant $v = h(w)$ to be used in the sub-models $\mathsf{M_1}$ and $\mathsf{M_2}$. What is the shape of h? Furthermore, when refining $\mathsf{M_2}$ we have to think about the necessary changes to $\mathsf{M_1}$. As a consequence of the current situation, interfaces are refined to implementation level before decomposition. This complicates the use of decomposition on higher levels of abstraction. We would prefer a method where the necessary reasoning can be restricted to one place. The functional invariant h should be evident and easily maintainable also in the face of potential changes to the sub-models and the interfaces.

Using our approach of interface instantiation this can be done. Because we are treating instantiation as a special form of refinement, we can combine interface instantiation steps with refinement steps. This gives us some liberty in arranging complex refinements. We also encourage a decomposition style where a separate theory of interface instantiation is maintained. We think that this contributes substantially to obtain models that are easier to understand and to modify. Interface instantiation supports a more incremental approach to decomposition because modifications that concern several components can be confined to only one place: the interface.

We call the very specific form of interface refinement that we use *interface* *instantiation.* To be useful, it should

(i) be more liberal than [1] while not increasing the proof effort,
(ii) help to structure complex mixtures of decomposition and refinement,
(iii) work seamlessly with Event-B as it is. (It should not depend on translations.)

We argue by means of a case study that we have achieved this. The case study addresses a difficulty of relating Event-B refinement to Problem Frames elaboration [12] discussed in [7]. It has been composed from [7] and [15]. We have down-sized it in order to focus on the problem of the refinement of external variables, that is, the interfaces. We have a tool for decomposition [18] but we do not have implemented a software tool for interface instantiation. Instantiation of carrier sets has been implemented similarly internally in the ProB tool [14], in order to achieve better performance when model checking and constraint checking [7]. The case study as presented in [15] uses Problem Frames to achieve traceability of requirements. We have not used Problem Frames in this article because they are not required to explain interface instantiation. This also permits us to cast the problem entirely in Event-B terminology. However, the proposed method of instantiation could be used with Problem Frames as employed in [7, 15]. This work extends prior work presented in [9] by allowing instantiations in a lattice of interfaces.

In the modularisation approach for Event-B presented in [11], the notion of interface has been used to capture software specifications using some interface variables and operations acting on these variables. The intention behind the use of interfaces is to separate specifications from their implementations. Our notion of interface is intended to provide efficient support for refining external variables following Abrial's decomposition method for system models. There was an earlier attempt at external variable refinement that is hinted at in the specification of the proof obligation generator for the Rodin tool [8]. This was considered too complicated and not feasible for large systems that are decomposed and refined repeatedly. We think, that our approach solves the problem. Poppleton [16] discusses external refinement based on Abrial's approach but also does not provide a practicable technique for doing so. The approach of modelling extensible records [6] also permits a form interface instantiation. A difficulty with using this approach is caused by the explicit mathematical model used for record representations and the need to specify always values for all fields of a record. However, extensible records could be used with our approach where it would appear useful. Behavioural interface refinement such as discussed in [17] addresses changing traces sub-models can exhibit, usually adding new events. It does not consider refinement of shared variables.

2. Event-B

Event-B models are described in terms of the two basic constructs: *contexts* and *machines*. Contexts contain the static part of a model whereas machines contain the dynamic part. Contexts may contain *carrier sets*, *constants*, *axioms*, and *theorems*, where carrier sets are similar to types [2]. A context D

may *extend* a collection of contexts $\mathsf{D_1}, \ldots, \mathsf{D_n}$. In this relationship we call D the *concrete* context and $\mathsf{D_1}, \ldots, \mathsf{D_n}$ the *abstract* contexts. Machines provide behavioural properties of Event-B models. Machines may contain *variables*, *invariants*, *theorems*, and *events*. Variables v describe the state of a machine. They are constrained by invariants $I(v)$. Theorems $L(v)$ describe consequences of the invariants, i.e., we have to prove $I(v) \Rightarrow L(v)$.

Events. Possible state changes (from v to v') are described by means of events. Each event is composed of a *guard* $G(t, v)$ and an *action* $x :| \, S(t, x, v')$, where t are *parameters* the event may contain and x are some variables (a subset of v). We denote an event e by

$$\mathsf{e} \mathrel{\widehat{=}} \mathsf{any}\ t\ \mathsf{where}\ G(t, v)\ \mathsf{then}\ x :| \ S(t, v, x')\ \mathsf{end}\ .$$

Nondeterministic action assigning x to be an element of a set $E(t, v)$ is denoted by $x :\in E(t, v)$. Deterministic actions are denoted by $x := E(t, v)$. We denote an event without parameters by

$$\mathsf{e} \mathrel{\widehat{=}} \mathsf{when}\ G(v)\ \mathsf{then}\ x :| \ S(v, x')\ \mathsf{end}\ ,$$

and an event without parameters and guard by

$$\mathsf{e} \mathrel{\widehat{=}} \mathsf{begin}\ x :| \ S(v, x')\ \mathsf{end}\ .$$

A special init event without parameters and guard is used for the initialisation.

Example 1 (Model)**.** We specify a system where a sender sends a set of messages to a receiver. In our model the sender keeps its messages in a variable s and the receiver keeps the received messages in a variable r. To guarantee that only messages can be received that could have been sent, we require $r \subseteq s$. The messages are described abstractly in context $\mathsf{msg0}$. The set of all messages in partioned into proper messages MSG and acknowledgements ACK:

```
context msg0
  constants $MSG, ACK, S$
  sets $M$
  axioms
    $MSG \neq \varnothing \land ACK \neq \varnothing \land M = MSG \cup ACK \land MSG \cap ACK = \varnothing$
    $S \subseteq MSG$
  end ,
```

and the behaviour of the system is described by machine $\mathsf{sere0}$,

```
machine sere0 sees msg0
  variables $s, r$
  invariants $s \subseteq MSG \land r \subseteq s$
  events
    event init $\widehat{=}$ begin $s, r := S, \varnothing$ end
    event trans $\widehat{=}$ any $x$ where $x \in s \setminus r$ then $r := r \cup \{x\}$ end
  end .
```

Message transfer is captured in one-shot with event trans. It is common in Event-B to start with a model as simple as this one and gradually add more detail to it by refinement [2].

Refinement. A machine N can refine another machine M. We call M the *abstract* machine and N the *concrete* machine. The state of the abstract machine is related to the state of the concrete machine by a *gluing invariant* $J(v, w)$ associated with the concrete machine N, where v are the variables of the abstract machine and w the variables of the concrete machine. Each event e of the abstract machine is *refined* by one or more concrete events f.

Example 2 (Refinement)**.** We prepare the decomposition of the model into a sender and a receiver by refining machine sere0 such that sender snd and receiver rec exchange messages by means of a channel m. In the refined model the sender deletes messages from its buffer u after they have been sent. The sender and the receiver implement a simple hand-shaking protocol based on the two kinds of messages MSG and ACK. We extend msg0 with two constants a and ma,

```
context msg1 extends msg0
  constants $a, ma$
  axioms
    $ma \in MSG \to ACK$
    $a \in ACK$
  end .
```

Later, after decomposing the system into the sender and the receiver, these constants will allow us to refine behaviour that was underspecified in the abstraction. Effectively, we have moved the corresponding nondeterminism into a context. As a result, we can share their instantiation among the components and refine them consistently. (Incidentally, we already have follwed this approach when introducing S in msg0.) The behaviour of the refined system is described by machine sere1,

```
machine sere1 refines sere0 sees msg1
  variables $u, r, m$
  invariants $u \subseteq s \wedge (\{m\} \cap MSG) \subseteq s \wedge u \subseteq MSG \wedge r \subseteq MSG \wedge m \in M$
  events
    event init $\widehat{=}$ begin $u, r, m := S, \varnothing, a$ end
    event snd $\widehat{=}$ any $x$ where $x \in u \wedge m \in ACK$ then $m, u := x, u \setminus \{x\}$ end
    event rec refines trans $\widehat{=}$ when $m \notin r \cup ACK$
                                  then $r, m := r \cup \{m\}, ma(m)$ end
  end .
```

We have introduced the redundant invariants $u \subseteq MSG$, $r \subseteq MSG$ and $m \in M$ to show where different invariants go when decomposing a machine. It may also be seen as a preparatory step for the decomposition permitting us to distribute some invariants of sere1 across the components after decomposing the machine.

Decomposition. The idea of decomposition [1, 4, 10] is to split a large model into smaller sub-models which can be handled more comfortably than the whole: one should be able to refine these sub-models independently. For the purpose of this article we limit the discussion to decomposition into two machines. Decomposition of a model M separates this model into sub-models M_1 and M_5. These sub-models can then be refined independently into machines N_1 and N_5. The correctness of the decomposition technique guarantees that the model N, obtained by recomposing N_1 and N_5, is a refinement of the original model M. Decomposition and recomposition are illustrated by Fig. 2. Note that recomposition is never explicitly carried out. It is only in principle possible. The central concern during modelling is to decompose an abstract machine and, subsequently, refine the different sub-models separately.

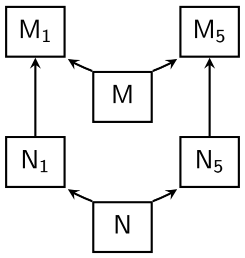

Figure 2: Decomposition and Recomposition

Let M be a machine with variables x_1, x_3, x_5 and invariants $I(x_1, x_3, x_5)$, $I_1(x_1, x_3)$, $I_3(x_3)$ and $I_5(x_3, x_5)$. Furthermore, let e_1, e_2, e_4 and e_5 be events of M, accessing different sets of variables as follows. Let

$\mathsf{e}_1 \mathrel{\widehat{=}} \mathsf{any}\ u_1\ \mathsf{where}\ E_1(u_1, x_1)\ \mathsf{then}\ x_1 :\mid P_1(u_1, x_1, x_1')\ \mathsf{end}$,
$\mathsf{e}_2 \mathrel{\widehat{=}} \mathsf{any}\ u_2\ \mathsf{where}\ E_2(u_2, x_1, x_3)\ \mathsf{then}\ x_1, x_3 :\mid P_2(u_2, x_1, x_3, x_1', x_3')\ \mathsf{end}$,
$\mathsf{e}_4 \mathrel{\widehat{=}} \mathsf{any}\ u_4\ \mathsf{where}\ E_4(u_4, x_3, x_5)\ \mathsf{then}\ x_3, x_5 :\mid P_4(u_4, x_3, x_5, x_3', x_5')\ \mathsf{end}$,
$\mathsf{e}_5 \mathrel{\widehat{=}} \mathsf{any}\ u_5\ \mathsf{where}\ E_5(u_5, x_5)\ \mathsf{then}\ x_5 :\mid P(u_5, x_5, x_5')\ \mathsf{end}$.

Machine M can be decomposed into two separate machines: M_1 with events e_1 and e_2; and M_5 with events e_4 and e_5. This is illustrated in Fig. 3. As a result of

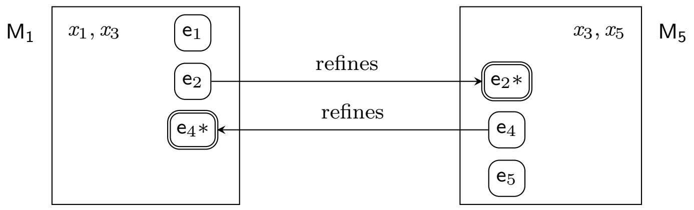

Figure 3: Maintaining the external invariant of several sub-models

the decomposition, M_1 has private variables x_1 and shared variables x_3. Invariants $I_1(x_1, x_3)$ and $I_3(x_3)$ can be attached to M_1. The resulting sub-machine M_1

has two *internal* events e_1 and e_2 and one *external* event e_4* which abstracts[1] e_4 projected on the state containing only x_1 and x_3, that is, e_4* abstracts the event any u_4, x_5 where $E_4(u_4, x_3, x_5)$ then $x_3 :| \exists x_5' \cdot P_4(u_4, x_3, x_5, x_3', x_5')$ end. Machine M_5 is similar to M_1, with the two internal events e_4 and e_5 and an external event e_2* that abstracts e_2 projected on x_3 and x_5. It has the private variables x_5 and the shared variables x_3. Machines M_1 and M_5 can be developed independently with the syntactical constraints that the shared variables cannot be removed and the external events can only be refined in a restricted way.

Note that invariant $I(x_1, x_3, x_5)$ is not copied to either M_1 or M_5. A possibility to use this invariant in one of the sub-models is to project also this invariant on the corresponding state using existential quantifier. Thus, $\exists x_5 \cdot I(x_1, x_3, x_5)$ could be added as an invariant to M_1.

Example 3 (Decomposition)**.** We decompose the machine sere1 into two machines sender1a and receiver1a. The sender has an internal event *snd* and an external event *rec*. We mark external events with an asterisk "$*$".

```
machine sender1a sees msg1
  variables $u, m$
  invariants $u \subseteq MSG \land m \in M$
  events
    event init $\widehat{=}$ begin $u, m := S, a$ end
    event snd $\widehat{=}$ any $x$ where $x \in u \land m \in ACK$ then $m, u := x, u \setminus \{x\}$ end
    event rec$*$ $\widehat{=}$ when $m \notin ACK$ then $m := ma(m)$ end
  end .
```

The event snd is a textual copy of the corresponding event from sere1, whereas rec$*$ is an abstraction of the corresponding event rec from sere1. (Variable r has been abstracted away. See the aside at the end of this example.) Thanks to the function ma event rec$*$ is deterministic and assigns an identical value to m as does the internal event rec of the receiver below.

```
machine receiver1a sees msg1
  variables $r\ m$
  invariants $r \subseteq MSG \land m \in M$
  events
    event init $\widehat{=}$ begin $r, m := \varnothing, a$ end
    event snd$*$ $\widehat{=}$ when $m \in ACK$ then $m :\in MSG$ end
    event rec $\widehat{=}$ when $m \notin r \cup ACK$ then $r, m := r \cup \{m\}, ma(a)$ end
  end .
```

Note that the assignment to m in the external event snd$*$ has become nondeterministic as a consequence of the abstraction from u.
Aside. In principle, decomposition can be done by a decomposition tool such as [18]. The external event rec$*$ as produced by the decomposition tool would

[1] "$\mathsf{e}*$ abstracts e" is the same as "e refines $\mathsf{e}*$".

have the following shape:

$$\mathsf{rec}{*} \mathrel{\widehat{=}} \mathsf{any}\ r\ \mathsf{where}\ r \subseteq M \wedge m \notin r \cup ACK\ \mathsf{then}\ m := ma(m)\ \mathsf{end}\ \ .$$

The tool automatically quantifies over the variables that are internal to another machine, in this case r. Manual decomposition usually leads to clearer models, however, at the cost of having to "invent" suitable abstractions.

3. Instantiation

Carrier set and constant instantiation. Contexts can be extended as usual in Event-B but we allow additionally to specify expressions to *instantiate* constants and carriers sets. Abstract carriers sets s must be instantiated by type expressions $e(t)$ and constants c can be instantiated by expressions $f(t, d)$, i.e.,

```
context C                    context D extends C with $s = e(t)$ , $c = f(t,d)$
  sets $s$                     sets $t$
  constants $c$                constants $d$
  axioms $A(s,c)$              axioms $B(t,d)$
  end                          end .
```

The equalities specifying the instantiation are treated similarly to axioms. The abstract constants and carriers sets that are instantiated remain visible. By contrast, the instantiation proposed in [2] replaces constants and carrier sets in the instantiating context. Still, they are similar to [2]: The equations of the extends-clause are used to rewrite the abstract axioms. If this changes an axiom, that axiom must be *proved* to hold in the instantiating context. Otherwise, nothing needs to be proved. This ensures that instantiation itself does not introduce new facts. The *instantiation* proof obligation is

$$B(t,d) \ \Rightarrow\ A(e(t), f(t,d))\ \ .$$

In summary, conventional Event-B context extension is instantiation with identity. Only abstract axioms of C with instantiated constants need to be proved as theorems in D. The other axioms are preserved by extension.

Connecting machines to interfaces. Interfaces are declared in contexts and used in machines by connecting a machine to the interface. The machines must see the corresponding context:

```
context C                          machine M
  interface U                        sees C
    fields $m$                       connects U
    constraints $P(m)$               ⋮
  ⋮                                  end .
  end
```

The constraints of an interface can refer to all constants and carrier sets of the surrounding context. In machine M the fields m are treated like variables and the constraints $P(m)$ like external invariants.

Example 4 (Interface)**.** The two machines sender1a and receiver1a share the external variable m. This variable plus the constraint $m \in M$ constitutes the interface bwetween the machines. We state this formally in a context msg2 that extends the context msg1,

```
context msg2 extends msg1
  interface itf2
    fields $m$
    constraints $m \in M$
  end .
```

Instead of sharing the variable m, the two machines are connected by interface itf2 replacing the variable m and invariant $m \in M$ in each machine,

```
machine sender1b sees msg2
  connects itf2
  variables $u$
  invariants $u \subseteq MSG$
  ⋮
  end
```

```
machine receiver1b sees msg2
  connects itf2
  variables $r$
  invariants $r \subseteq MSG$
  ⋮
  end .
```

Modifications to the interface of the two machines can now be carried out in one place, namely, the interface itf2.

Interface instantiation. Interfaces can be instantiated by specifying equalities $m = h(n)$ for replacing fields of an abstract interface m by fields of a concrete interface n. The names on the right-hand side of the equation must not occur in the abstract interface. Let interface V be given by

```
context D extends C
  interface V instantiates U with $m = h(n)$
    fields $n$
    constraints $Q(n)$
  end .
```

The expression h is often composed of pair-expressions "$\cdot \mapsto \cdot$". Interface instantiations must be bijective: we have to prove that h^{-1} is a function. The constraints $P(m)$ of U are contained in interface V as specified by the instantiation $m = h(n)$, that is, they become $P(h(n))$. Similarly to machine variables, field names of interfaces cannot be reintroduced. Similarly to machine invariants, constraints are accumulated in by instantiation: the constraints of interface V are $Q(n) \wedge P(h(n))$. Fig. 4 illustrates the interaction between decomposition, interface instantiation and refinement.

Example 5 (Instantiation)**.** The implementation of the sender-receiver system distinguishes proper messages from acknowledgements by a Boolean value. Messages are thus composed of an ID, the Boolean value indicating whether the message is proper or an acknowledgement, and the message contents. Formally,

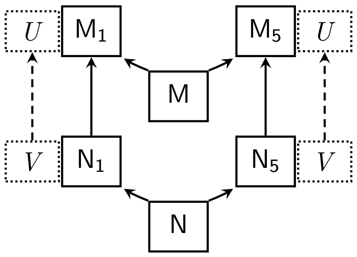

Figure 4: Interface instantiation

the set of messages M is instantiated by $ID \times \mathbb{B} \times C$ and the other constants accordingly,

```
context msg3 extends msg2 with
        $M = ID \times \mathbb{B} \times C$
        $MSG = ID \times \{\mathbf{F}\} \times C$
        $ACK = ID \times \{\mathbf{T}\} \times C$
        $a = b \mapsto \mathbf{T} \mapsto h$
        $ma = (\lambda j, f, d \cdot j \in ID \wedge f = \mathbf{F} \wedge d \in C \mid j \mapsto \mathbf{T} \mapsto d)$
    sets $ID, C$
    constants $b, h$
    axioms $b \in ID \wedge h \in C$
    interface itf3 instantiates itf2 with $m = i \mapsto t \mapsto c$
        fields $i, t, c$
        constraints $i \in ID \wedge t \in \mathbb{B} \wedge c \in C$
    end .
```

Note that the constant S is not instantiated. It is not required that all carrier sets and constants be instantiated. The need for instantiation is usually determined by the refinement proofs in the connected machines. We have to prove that the instantiation of the carrier sets and constants M, MSG, ACK, a and ma is consistent, that is

$$(ID \times \{\mathbf{F}\} \times C) \cup (ID \times \{\mathbf{T}\} \times C) = ID \times \mathbb{B} \times C \quad ,$$
$$(ID \times \{\mathbf{F}\} \times C) \cap (ID \times \{\mathbf{T}\} \times C) = \varnothing \quad ,$$
$$(\lambda j, f, d \cdot j \in ID \wedge f = \mathbf{F} \wedge d \in C \mid j \mapsto \mathbf{T} \mapsto d) \in (ID \times \{\mathbf{F}\} \times C) \to (ID \times \{\mathbf{T}\} \times C) \quad ,$$
$$b \mapsto \mathbf{T} \mapsto h \in ID \times \{\mathbf{T}\} \times C \quad .$$

Constraints of interfaces are only accumulated, hence, nothing needs to be proved about the instantiated interface.

Interface instantiation lattice. An interface V may also instantiate several interfaces $\mathsf{U}_1, \ldots, \mathsf{U}_k$. This is only possible under the following condition that must be satisfied in V for all fields m of V or some abstract interface: *The fields instantiated in distinct interfaces U_ℓ and U_r are distinct or have been instantiated* *in a common abstract interface.* (A similar restriction is used in Event-B for context extension to ensure that carrier sets and constants have unique declarations.) As a consequence of the instantiation condition we only need to verify the following for all fields m instantiated in V with $m = h_0(n)$ and in U_ℓ with $m = h_\ell(n)$: $h_0(n) = h_\ell(n)$.

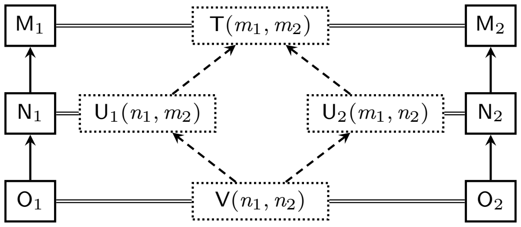

Figure 5: A lattice of interfaces

A development using interfaces may have different branches of sub-models with distinct interfaces that are joined ultimately so that all machines of a model agree on their interfaces. We illustrate this by way of the small example shown in Fig. 5. In the figure, an abstract interface T containing two abstract fields m_1 and m_2 is used by M_1 and M_2. The development of sub-models M_1 and M_2 instantiates the fields of the interfaces in different order. Interface T is instantiated by U_1 where m_1 is replaced by n_1 and m_2 is retained while M_1 is refined into N_1 using U_1. Similarly, M_2 is refined by N_2 using U_2 that replaces m_2 by n_2 and retains m_1. Finally, V instantiates the two interfaces U_1 and U_2. The new interface V is used to refine N_1 and N_2 into O_1 and O_2.

Such a lattice of interfaces allows us to vary the order in which different fields are instantiated in different branches of a development. In the course of this, we temporarily abandon the compositionality of the intermediate machines (here N_1 and N_2), only to reestablish it later for the final machines O_1 and O_2, by connecting them to the same interface V.

External event refinement. Using interface instantiation we permit refinement of external events. Consider the following external event e operating on the external variables x and its refinement f operating on the external variables y. The refinement of external variables is captured by the relationship of the form $x = h(y)$. Note that external events do not refer to any internal variables: they can only refer to external variables of the corresponding model. Let

$$\begin{aligned}
&\mathsf{e} \;\widehat{=}\; \mathsf{any}\ u\ \mathsf{when}\ E(u,x)\ \mathsf{then}\ x :\mid P(u,x,x')\ \mathsf{end}\ ,\\
&\mathsf{f} \;\widehat{=}\; \mathsf{any}\ v\ \mathsf{when}\ F(v,y)\ \mathsf{with}\ W(u,v,x,y,y') \wedge x' = h(y')\ \mathsf{then}\ y :\mid Q(v,y,y')\ \mathsf{end}\ ,
\end{aligned}$$

where $W(u,v,x,y,y') \wedge x' = h(y')$ is the witness for the refinement of e by f. The witness incorporates the refinement of external variables with function h.

Beside the proof obligations to prove that f is a refinement of e, we also need to prove that f is refined by e. The latter is proved using the same given witnesses by the following proof obligations. We abbreviate the hypothesis common to proof obligations by

$$L(x, y, u, x, x') = I(x) \land J(x, y) \land x = h(y) \land E(u, x) \land P(u, x, x') \enspace .$$

We have to prove *witness feasibility,*

$$L(x, y, u, x, x') \Rightarrow (\exists v, y' \cdot W(u, v, x, y, y') \land x' = h(y')) \enspace .$$

Because h is bijective, the existence of y' is trivial, and the proof obligation can be simplified to $L(x, y, u, x, x') \land x' = h(y') \Rightarrow (\exists v \cdot W(u, v, x, y, y'))$. We have to show that concrete external events display the same behaviour as abstract events, that is, that nondeterminism is not reduced. We prove *guard weakening,*

$$L(x, y, u, x, x') \land W(u, v, x, y, y') \land x' = h(y') \Rightarrow F(v, y) \enspace ,$$

and *(co-)simulation*,

$$L(x, y, u, x, x') \land W(u, v, x, y, y') \land x' = h(y') \Rightarrow Q(v, y, y') \enspace .$$

Note that *invariant preservation* for the refinement of f by e can be derived from the *invariant preservation* for the refinement of e by f and the fact that we use the same witnesses.

Example 6 (Instantiation and refinement)**.** The instantiated interface itf3 is incorporated into the model by refining the machines sender1b and receiver1b connecting the refinements to the instantiated interfaces. In the refinement proof all equalities specified in msg3 and itf3 can be used. The sender is refined by the machine

```
machine sender2b refines sender1b sees msg3
  connects itf3
  variables $u$
  events
    event init $\widehat{=}$ begin $u, i, t, c := S, b, \mathbf{T}, h$ end
    event snd $\widehat{=}$ any $j, f, d$ where $j \mapsto f \mapsto d \in u \land t = \mathbf{T}$
                  then $i, t, c, u := j, f, d, u \setminus \{j \mapsto f \mapsto d\}$ end
    event rec$*$ $\widehat{=}$ when $t = \mathbf{F}$ then $t := \mathbf{T}$ end
  end ,
```

and the receiver by the machine

```
machine receiver2b refines receiver1b sees msg3
  connects itf3
  variables $r$
  events
    event init $\widehat{=}$ begin $r, i, t, c := \varnothing, b, \mathbf{T}, h$ end
    event snd$*$ $\widehat{=}$ when $t = \mathbf{T}$ then $i \mapsto t \mapsto c :\in MSG$ end
    event rec $\widehat{=}$ when $i \mapsto t \mapsto c \notin r \land t = \mathbf{F}$
                  then $r, t := r \cup \{i \mapsto t \mapsto c\}, \mathbf{T}$ end
  end .
```

We must prove the simulation relationships between the events of sender2b and receiver2b and those of sender1b and receiver1b. Concerning the interface itf3, we have to show that its constraints are preserved by all events. Because the constraints of itf3 are equivalent to *true*, the proof is trivial.

Example 7 (Emulation of instantiation using bijections)**.** Two tools are available for decomposition [18] and instantiation [5]. However, they transform Event-B machines syntactically producing component machines and instantiaed machines that do not have a relationship to their abstractions or among each other. This approach makes it difficult to maintain complex models. Our approach improves the situation by offering interfaces and instantiation supported by the modelling formalism. Not having dedicated tool support for the instantiation method yet, we emulate the method by specifying bijections that model the equations between instantiated terms,

```
context msg2x extends msg1
  sets $ID, C$
  constants $U, na, b, h, \iota$
  axioms
    $\iota \in ID \times \mathbb{B} \times C \rightarrowtail\!\!\!\!\twoheadrightarrow M$
    $\iota^{-1}[MSG] = ID \times \{\mathbf{F}\} \times C$
    $\iota^{-1}[ACK] = ID \times \{\mathbf{T}\} \times C$
    $U = \iota^{-1}[S]$
    $na = (\lambda j, f, d \cdot j \in ID \wedge f = \mathbf{F} \wedge d \in C \mid j \mapsto \mathbf{T} \mapsto d)$
    $ma = \iota^{-1};na;\iota$
    $b \mapsto \mathbf{T} \mapsto h = \iota^{-1}(a)$
  end .
```

The most elaborate proof of this development using the emulation occurs when instantiating function ma. With instantiation support in place this would have been trivial using the fact that $ma = \ldots$ as specified in context msg3. The difficulty in the proof of refinement emulating instantiation is caused by the need to use the bijection ι so that the equation for the instantiation becomes $ma = \iota^{-1};na;\iota$. Similar complications occur for all other abstract constants x (and sets X) where the equations have the form $x = \iota(y)$ (and $X = \iota[Y]$). In the refined machines the instantiation equations need to be stated in the invariants for the sender

```
machine sender2a refines sender1a sees msg2x
  variables $v, i, t, c$
  invariants $u = \iota[v] \wedge m = \iota(i \mapsto t \mapsto c)$
  events
    event init $\widehat{=}$ begin $v, i, t, c := U, b, \mathbf{T}, h$ end
    event snd $\widehat{=}$ any $j, f, d$ where $j \mapsto f \mapsto d \in v \wedge t = \mathbf{T}$
                    then $i, t, c, v := j, f, d, v \setminus \{j \mapsto f \mapsto d\}$ end
    event rec$*$ $\widehat{=}$ when $t = \mathbf{F}$ then $t := \mathbf{T}$ end
  end
```

and the receiver

```
machine receiver2a refines receiver1a sees msg2x
   variables $q, i, t, c$
   invariants $r = \iota[q] \wedge m = \iota(i \mapsto t \mapsto c)$
   events
      event init $\widehat{=}$ begin $q, i, t, c := \varnothing, b, \mathbf{T}, h$ end
      event snd$*$ $\widehat{=}$ when $t = \mathbf{T}$ then $i \mapsto t \mapsto c :\in MSG$ end
      event rec  $\widehat{=}$  when $i \mapsto t \mapsto c \notin q \wedge t = \mathbf{F}$
                       then $q, t := q \cup \{i \mapsto t \mapsto c\}, \mathbf{T}$ end
   end .
```

Finally, we want to point out that in context msg2x consistency between the different axioms is not assured. It is easily possible to specify a collection of inconsistent axioms and techniques are required to validate them as can be done, for instance, with the ProB tool [14].

4. Case study: modelling of a cruise control system

We present interface instantiation by means of a model of a cruise control system. The cruise control system permits the driver of a car to select a target speed that the vehicle should attain. The system will try to maintain the vehicle speed as close as possible to the target speed. Since our main interest is to discuss interface instantiation, we will only discuss the functionality of the cruise control system as far as necessary for that discussion. We have modelled the system using the Rodin tool [3], emulating instantiation similarly to the approach of [7]: interfaces are represented syntactically by a lexical convention and carrier set instantiation is modelled by suitable bijections. The purpose of this model is to check the consistency of our cruise control model by way of a similar model verified using the Rodin tool. It is difficult to draw conclusions about the power or ease of the proof method based on that formal model. The reason for this is a mismatch of the proof obligations between our approach and what can be done with the tool currently. For this case study this is acceptable because as with many industrial applications of formal methods the challenge of the original problem is in the size of the model and not in the difficulty of the specified formal properties.

4.1. Development strategy

We want to implement a cruise control system sy0 by three sub-models: the controller cr5, the engine en5 and the exterior ae1. Fig. 6 shows the sub-models and their interfaces. The implementations of the controller and the engine are connected by means of two interfaces: concrete speed and acceleration, psa, and concrete pedal signals translated and passed on from the exterior, psi. The interface to the exterior, aes, is kept abstract in the implementation. More abstract system models should not be forced to use the interfaces psa and psi but permit abstractions thereof as shown in Fig. 7. The details of the interfaces

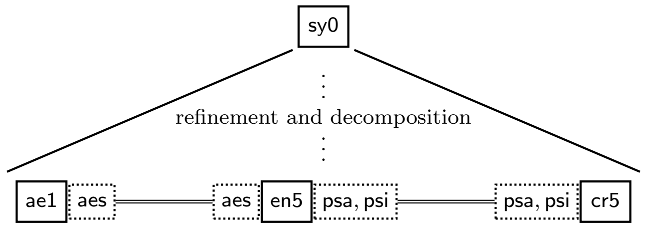

Figure 6: Architecture of the system in terms of sub-models and interfaces

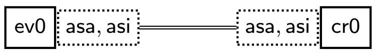

Figure 7: Abstract sub-models and their abstract interfaces

should be introduced step by step, introducing the abstract interfaces asa and asi first. We prefer to refine the controller and the engine but keep the exterior abstract at first. We do not want to decide on all interfaces before decomposing system sy0: we have not decided yet on the shape of the implementation of sub-model ae1 and of interface aes. Interface aes could be used to implement an interface to the exterior or it could be used for animation and visualisation [13], for instance. The problem we face is to fit the abstract sub-models of Fig. 7 between sy0 and the implementation in Fig. 6.

4.2. The full model: refinement, decomposition and instantiation

We present an overview of the full model and discuss specific issues in subsequent sections. Fig. 8 shows the details of the development outlined in Fig. 6 with the abstract decomposed model of Fig. 7 incorporated at the top of the figure. The contexts ctx0, ctx1, ctx2 and ctx3 specify the interface instantiations as indicated on the right-hand side of the figure and accompanying instantiations of carrier sets and constants. Context extension is shown on the left-hand side of the figure. Machines that see a context are depicted in the box of the corresponding context. For instance, the abstract model sy0 of the cruise control is shown in the box of context ctx0. Most of the development effort focuses in the two columns en1 to en5 (on the engine) and cr1 to cr5 (on the controller).

We do not discuss all aspects of the development but focus on the following four aspects of the engine and the controller development.

i. Interface introduction (Section 4.3): In order to decompose a machine some variables private to that machine become shared among the sub-machines. For these variables interfaces are specified and the sub-machines are connected to them.
ii. Mixing instantiation and refinement (Section 4.4): Usually, a change of the representation of shared variables needs to be accompanied by refinements of private variables. Furthermore, carrier set and constant instantiation can be used to support refinement of private variables.

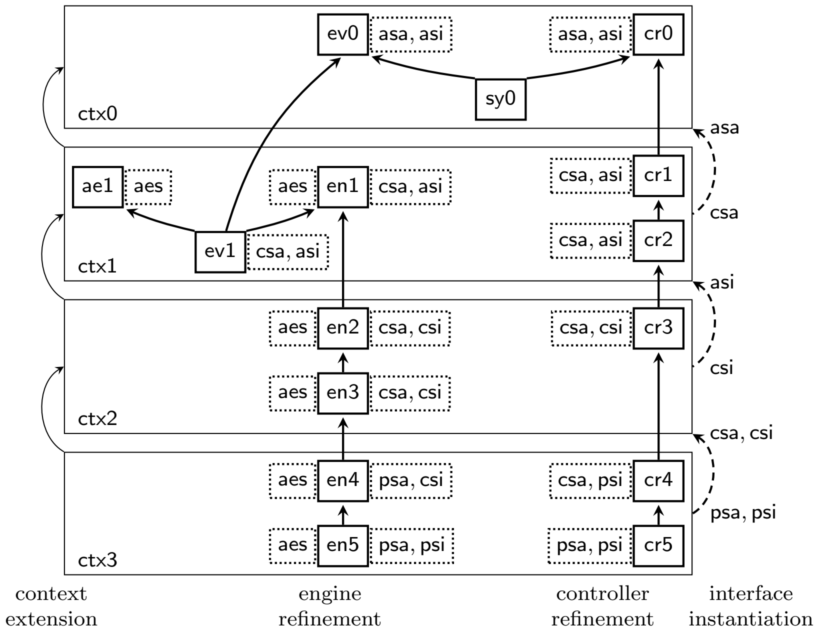

Figure 8: Overview of the cruise control system model

iii. Repeated instantiation (Section 4.5): Details about interfaces are introduced incrementally. Hence, it is important that the process of instantiation can be repeated without difficulty.
iv. Instantiation branching (Section 4.6): If two branches in a development instantiate interface fields in different order, then there must be a common instantiation that permits recomposition of the sub-models.

4.3. Interface introduction

The abstract model. The model sy0 from which we start the development declares variables *sig*, *cs*, *vs*, *md*, *ts*, *va* modelling external signals *sig*, internal control signals *cs*, vehicle speed *vs*, control mode *md*, target speed *ts* and acceleration *va*. It does not connect to any interfaces. This means we can refine this model in the usual way. Context ctx0 declares constants *ES*, *CS*, *VS*, *VA*, *VRA*, etc., modelling external signals *ES*, control signals *CS*, vehicle speed *VS*, vehicle acceleration *VA*, restricted vehicle acceleration *VRA*. It postulates the axiom

$$VRA \subseteq VA \ . \tag{1}$$

We have invented constant *VRA* to make the invariant more interesting. The constants determine the possible values of the variables by means of the invariant of sy0:

$$sig \in ES \wedge cs \in CS \wedge \ldots \wedge md \in \{C, AC, NC\} \wedge (md = C \Rightarrow va \in VRA) \ .$$

The constant C models "cruise control active"; AC models "manual change of vehicle speed"; NC models "cruise control not active". Constants CS and VS are declared to be contained in carrier sets, for example, carrier set K contains CS and carrier set S contains VS. The carrier sets themselves are not used in the machine. The reason for this is that they can only be instantiated by type expressions. However, the more common case is that we need to instantiate by some more constrained set. See, for example, the instantiations of C, AC and NC in Section 4.4.

Events of the abstract model. We discuss three events of machine sy0: event chm ("change mode") models an internal state change of the controller,

$$\textsf{event chm} \mathrel{\widehat{=}} \textsf{begin}\ md :\mid md' \in \{C, AC, NC\} \wedge (md'{=}C \Rightarrow va \in VRA)\ \textsf{end} \ ;$$

event chaac ("change acceleration in mode AC") models output to the engine,

$$\textsf{event chaac} \mathrel{\widehat{=}} \textsf{when}\ md = AC\ \textsf{then}\ va :\in VA\ \textsf{end} \ ;$$

event chcs ("change control signals") models input from the engine,

$$\textsf{event chcs} \mathrel{\widehat{=}} \textsf{begin}\ cs := fcs(sig)\ \textsf{end} \ .$$

It would be tempting to specify in the abstract event chcs the assignment $cs := sig$. However, this asserts that cs and sig have the same type. Once the system is decomposed, we would have to refine them in the same way. To avoid this, we have introduced function fcs mapping from the type of sig to the type of cs. Models always need to be prepared for decomposition. Our method of instantiation does not change this.

Decomposition of the abstract model. Decomposing sy0 into ev0 and cr0 we have to introduce interfaces asa and asi:

```
interface asi                    interface asa
  fields $cs$                      fields $vs, va$
  constraints $cs \in CS$          constraints $vs \in VS \wedge va \in VA$ .
```

Machine ev0 has one internal variable sig and connects to the two interfaces asa and asi. Machine cr0 connects to the same interfaces and has two internal variables ts and md. We split the events in the usual way depending on which variables and fields the events refer to. Except for the use of interfaces the decomposition method of [1] works as before.

Mode	**Submode**	**Description**
NC	OFF	Ignition is on, cruise control initialised and switched off
	ERR	An irreversible error has occurred
	REC	A reversible error has occurred
C	CRS	Cruise control is maintaining the target speed
	RES	Target speed is approached from above or from below
AC	ACC	Cruise control is accelerating the car
	DEC	Cruise control is decelerating the car

Table 1: Modes and submodes of the cruise control system.[2]

4.4. Mixing instantiation and refinement

Instantiation. We refine cr0 to cr1 by instantiating interface asa by csa while refining variable md by variable nd. The constraints and instantiation equalities of csa become part of the gluing invariant of cr1. The abstract constants VS, VA and VRA are instantiated by integer ranges: $VS = mS \mathrel{..} MS$, $VA = mA \mathrel{..} MA$ and $VRA = mRA \mathrel{..} MRA$ constrained by axioms

$$\ldots \wedge 0 \in mRA \mathrel{..} MRA \wedge mA \leq mRA \wedge mRA \leq MRA \wedge MRA \leq MA \wedge \ldots \quad (2)$$

To satisfy the instantiation proof obligation we have to verify that (2) implies (1).

For clarity we introduce a new name for the interface containing the instantiated constants: interface csa instantiates asa. Machine cr1 and ev1 now both need to be connected to interface csa replacing asa. The machine also need to see the extended context ctx1.

Refinement. Variable md is refined by instantiating the constants C, AC and NC, using the gluing invariant $nd \in md$, and constant instantiations $C = \{CRS, RES\}$, $AC = \{ACC, DEC\}$, $NC = \{OFF, ERR, REC\}$. Table 1 gives an overview of the operation modes and submodes of the model of the cruise control system. Note, how closely constant instantiation and refinement are linked in the refinement of md. The type of the abstract variable md has been instantiated such that the gluing invariant simply becomes $nd \in md$.

4.5. Repeated instantiation

First instantiation. Continuing the development from cr1 and ev1, we first instantiate interface asi by csi

```
interface csi instantiates asi with $cs = (ps \mapsto cis \mapsto is)$
  fields $ps, cis, is$
  constraints $ps \in PS \wedge cis \in CIS \wedge is \in IS$
```

[2]Table adapted from [7].

Pedal Signal	**Description**
pbp	The brake pedal has been hit
pbe	An error has occured in the brake subsystem
pcp	The clutch pedal has been hit
pce	An error has occured in the clutch subsystem
pae	An error has occurred in the accelerator subsystem

Table 2: Concrete pedal signals of the cruise control system.

where PS, for "pedal signals", is a constant of context ctx2. The context also declares two constants PSE and PSS, for "pedal signals error" and "pedal signals success", such that $PSE \subseteq PS \wedge PSS \subseteq PS \wedge PSE \cap PSS = \varnothing$. This is used for a first refinement of event chm into two events chme and chmn:

```
event chme refines chm $\widehat{=}$
  when $ps \in PSS \cup PSE$ then $nd :\in \{ERR, REC\}$ end
event chmn refines chm $\widehat{=}$
  when $ps \notin PSS \cup PSE$
  then $nd :| \; nd' \in \{CRS, RES\} \Rightarrow va \in mRA \,..\, MRA$ end .
```

Second instantiation. Subsequently we instantiate the interface csi by psi

```
interface psi instantiates csi with $ps = (pbp \mapsto pbe \mapsto pcp \mapsto pce \mapsto pae)$
  fields $pbp, pbe, pcp, pce, pae, cis, is$
  constraints $pbp \in \mathbb{B} \wedge pbe \in \mathbb{B} \wedge pcp \in \mathbb{B} \wedge pce \in \mathbb{B} \wedge pae \in \mathbb{B}$
```

representing the abstract pedal signals ps by a bit vector of concrete pedal signals $pbp \mapsto pbe \mapsto pcp \mapsto pce \mapsto pae$. Table 2 gives an overview of the different concrete pedal signals. We instantiate the constants PSE and PSS

$$\begin{aligned} PSE &= \{bp \mapsto be \mapsto cp \mapsto ce \mapsto ae | \mathbf{T} \in \{be, ce, ae\}\} \;, \\ PSS &= \{bp \mapsto be \mapsto cp \mapsto ce \mapsto ae | \mathbf{T} \notin \{be, ce, ae\} \wedge \mathbf{T} \in \{bp, cp\}\} \;. \end{aligned}$$

by and prove $PSE \cap PSS = \varnothing$ as postulated above.

Event chme is split (that is, refined into several events) according to different signal combinations.

```
event chmrb refines chme $\widehat{=}$
  when $\mathbf{T} \notin \{pbe, pce, pae\} \wedge \mathbf{T} = pbp$ then $nd := REC$ end
event chmrc refines chme $\widehat{=}$
  when $\mathbf{T} \notin \{pbe, pce, pae\} \wedge \mathbf{T} = pcp$ then $nd := REC$ end
event chmbe refines chme $\widehat{=}$ when $\mathbf{T} = pbe$ then $nd := ERR$ end
event chmce refines chme $\widehat{=}$ when $\mathbf{T} = pce$ then $nd := ERR$ end
event chmae refines chme $\widehat{=}$ when $\mathbf{T} = pae$ then $nd := ERR$ end .
```

This last instantiation is much more concise than the refinement suggested in [7]. We can avoid a lot of the overhead that is usually incurred by using refinement emulating instantiation.

4.6. Instantiation branching

In machine en3 event chvs is refined by replacing the abstract nondeterministic assignment $vs :\in mS \,..\, MS$ by a deterministic assignment using a function sf that describes engine acceleration and deceleration,

$$\textsf{event chvs refines chvs} \;\widehat{=}\; \textsf{begin}\; vs := sf(va \mapsto vs)\; \textsf{end}\;\;.$$

Field va of interface csa is instantiated as the difference between two fields vap and van in en4. The two fields model acceleration and decelaration explicitly by non-negative values. It is easy to verify that the new representation of the vehicle acceleration is unique. This is the corresponding interface instantiation:

$$\begin{array}{l}\textsf{interface psa instantiates csa with}\; va \;=\; vap{-}van\\ \quad\textsf{fields}\; vs, vap, van\\ \quad\textsf{constraints}\; vap \in 0 \,..\, MA \wedge van \in 0..{-}mA \wedge 0 \in \{vap, van\}\;\;.\end{array}$$

Event chvs needs to be refined in en4. The simplest refinement possible is obtained by replacing va by the difference $vap{-}van$:

$$\textsf{event chvs refines chvs} \;\widehat{=}\; \textsf{begin}\; vs := sf(vap{-}van \mapsto vs)\; \textsf{end}\;\;.$$

The two machines en4 and cr4 have different interfaces and cannot be composed. Both are instantiated in one more development step so that their interfaces coincide again and they are composable. The method of instantiation permits to apply instantiations in different orders or instantiate fields the same interface in different orders. Both approaches are based on the same theory. In this case study we have used the more explicit way of applying interfaces in different orders. It could easily be recasted into the second form of instantiation by combing the two interfaces asi and asa.

5. Correctness

We have used the Rodin tool [3] to verify the correctness of interface refinement. First we present a technique for verifying extensions of Event-B. We believe that it is useful beyond the use in this article for verifying the correctness of interface instantiation.

5.1. A technique for proving correctness of Event-B extensions

The general idea is to encode a generic model using the Rodin tool, and illustrating the extended method using the generic model. Typically, the correctness of an extension can be stated as follows: assume the consistency of some *input model*, then prove the consistency of the extended *output model*. The approach is illustrated in Fig. 9 and contains four steps as follows.

1. Encode the generic input model.

2. Encode the generic output model.

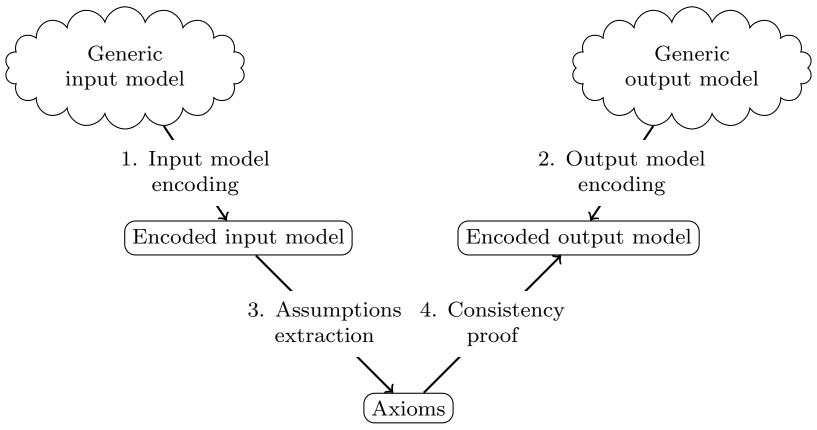

Figure 9: Approach for proving correctness of Event-B extensions

3. Gather the consistency conditions of the input model. The consistency of a model is described by the associated proof obligations. To turn them into assumptions, we make these proof obligations axioms.

4. Prove the consistency of the output model using these axioms.

To illustrate the approach, we prove that restricted superposition refinement preserves invariance. Our input model is an abstract machine M and its refinement N as illustrated in Fig. 10. Variables x is retained through the (superposition) refinement. Abstract event e and concrete event f have the same parameters u. Our output model is a flattened copy of N (without the refines clause) with an additional invariant $I(x)$.

```
machine M
variables $x$
invariants $I(x)$
events
  event e $\widehat{=}$
    any $u$ where
      $E(u, x)$
    then
      $x :\mid P(u, x, x')$
    end
end
```

```
machine N
refines M
variables $x$
events
  event f refines e $\widehat{=}$
    any $u$ where
      $F(u, x)$
    then
      $x :\mid Q(u, x, x')$
    end
end
```

Figure 10: Generic machines M and its refinement N

To encode the generic machine M, we first model the type of variables x and parameters u using some carrier sets X and U. Subsequently, I, E and P can be declared as constants with appropriate type, i.e. $I \in \mathbb{P}(X)$, $E \in \mathbb{P}(U \times X)$, and $P \in \mathbb{P}(U \times X \times X)$ as illustrated with context M_ctx in Fig. 11. The machine M is encoded accordingly using the above context, where predicates are translated using the set membership operator ($\in$).[3] For example, the invariant $I(x)$ is translated to $x \in I$. The encoded machine M_mch is in Fig. 11. Other machines

```
context M_ctx
sets $X, U$
constants $I, E, P$
axioms
  $I \in \mathbb{P}(X)$
  $E \in \mathbb{P}(U \times X)$
  $P \in \mathbb{P}(U \times X \times X)$
end
```

```
machine M_mch
sees M_$ctx$
variables $x$
invariants $x \in I$
events
  event e $\widehat{=}$
    any $u$ where
      $u \mapsto x \in E$
    then
      $x :\mid u \mapsto x \mapsto x' \in P$
    end
end
```

Figure 11: Rodin encoding of M

are encoded in Rodin similarly.

We assume that the input machines M and N have been proved, including the proof obligation stating that event e maintains invariant $I(x)$. To turn it into an assumption, we encode the obligation, i.e.,

$$I(x) \wedge E(u,x) \wedge P(u,x,x') \Rightarrow I(x') \qquad \text{(e/INV)}$$

as axiom e/INV in the extended context M_po (Fig. 12).

```
context M_po
extends M_ctx
axioms
    e/INV :  $\forall x, u, x' \cdot x \in I \;\wedge\; u \mapsto x \in E \;\wedge\; u \mapsto x \mapsto x' \in P \;\Rightarrow\; x' \in I$
    ...
end
```

Figure 12: Encoding proof obligations as axioms

Similarly, the fact that f is a correct refinement of e is captured by the *guard strengthening* and *simulation* proof obligations which are encoded as the following axioms:

$$\forall u, x \cdot x \in I \wedge u \mapsto x \in F \;\Rightarrow\; x \in E \;, \qquad \text{(f/GRD)}$$

and

$$\forall u, x, x' \cdot x \in I \wedge u \mapsto x \in E \wedge u \mapsto x \mapsto x' \in Q \Rightarrow u \mapsto x \mapsto x' \in P \;. \qquad \text{(f/SIM)}$$

We prove the consistency of the output machine (a flattened copy of N with invariant I), from the axioms collected previously. For example, the proof

[3]This is a common technique to model predicate constants or variables in Event-B using first-order logic.

obligation stating that I is maintained by the concrete event f is as follows:

$$x \in I \land u \mapsto x \in F \land u \mapsto x \mapsto x' \in Q \Rightarrow x' \in I \ . \qquad (\mathsf{f/INV})$$

It is easy to see that f/INV is a consequence of e/INV, f/GRD and f/SIM.

5.2. Correctness of interface instantiation

Using the above proof method, we prove the correctness of interface instantiation as follows. For clarity, we present our proofs in this section in its generic form instead of its Rodin encoding.

5.2.1. The input model

The first component of our input model is a machine M with variables x_1, x_2, x_3, and x_4. Furthermore, let e_1, e_2, e_4 and e_4 be events of M, accessing different sets of variables as follows,

$$\mathsf{e}_1 \mathrel{\widehat{=}} \mathsf{any}\ u_1\ \mathsf{where}\ E_1(u_1, x_1)\ \mathsf{then}\ x_1 :\mid P_1(u_1, x_1, x_1')\ \mathsf{end}\ ,$$

$$\begin{aligned} \mathsf{e}_2 \ \widehat{=}\ & \mathsf{any}\ u_2\ \mathsf{where}\ E_2(u_2, x_1, x_2, x_3)\ \mathsf{then} \\ & \quad x_1, x_2, x_3 :\mid P_2(u_2, x_1, x_2, x_3, x_1', x_2', x_3') \\ & \mathsf{end}\ , \end{aligned}$$

$$\begin{aligned} \mathsf{e}_3 \ \widehat{=}\ & \mathsf{any}\ u_3\ \mathsf{where}\ E_3(u_3, x_2, x_3, x_4)\ \mathsf{then} \\ & \quad x_2, x_3, x_4 :\mid P_3(u_3, x_2, x_3, x_4, x_2', x_3', x_4') \\ & \mathsf{end}\ , \end{aligned}$$

$$\mathsf{e}_4 \mathrel{\widehat{=}} \mathsf{any}\ u_4\ \mathsf{where}\ E_4(u_4, x_4)\ \mathsf{then}\ x_4 :\mid P(u_4, x_4, x_4')\ \mathsf{end}\ .$$

We assume that the invariants of M can be separated into: $I(x_1, x_2, x_3, x_4)$, $I_{123}(x_1, x_2, x_3)$, $I_{23}(x_2, x_3)$, $I_{234}(x_2, x_3, x_4)$.

Decomposition. We decompose M into M_{12} and M_{34}, sharing the interface U. The abstract interface U encapsulates the shared variables x_2 and x_3 with invariant $I_{23}(x_2, x_3)$,

$$\begin{array}{ll} \mathsf{interface} & \mathsf{U} \\ \quad \mathsf{fields} & x_2, x_3 \\ \quad \mathsf{constraints} & I_{23}(x_2, x_3)\ . \end{array}$$

Besides connecting to interface U, sub-machine M_{12} has a private variable x_1 and invariant $I_{123}(x_1, x_2, x_3)$. Furthermore, original events e_1 and e_2 are copied as internal event of M_{12}. An external event a_3 is generated from the original event e_3 as follows:

$$\begin{aligned} \mathsf{a}_3 \ \widehat{=}\ & \mathsf{any}\ u_3, x_4\ \mathsf{where}\ E_3(u_3, x_2, x_3, x_4)\ \mathsf{then} \\ & \quad x_2, x_3 :\mid \exists x_4' \cdot P_3(u_3, x_2, x_3, x_4, x_2', x_3', x_4') \\ & \mathsf{end}\ . \end{aligned}$$

Machine M_{34} has a similar structure to M_{12}.

Interface Instantiation and Refinement. Assume that we develop M_{12} and M_{34}, by instantiating the shared interface U according to Fig. 13, where the interfaces are developed as follows.

```
interface        V_12                      interface        V_34
  instantiates                               instantiates
    U   with     $x_2 = h_2(y_2)$              U   with     $x_3 = h_3(y_3)$
  fields         $y_2, x_3$                  fields         $x_2, y_3$
  constraints    $J_{12}(y_2, x_3)$  ,        constraints    $J_{34}(x_2, y_3)$  ,
```

```
interface        V
  instantiates
    V_12  with   $x_3 = h_3(y_3)$
    V_34  with   $x_2 = h_2(y_2)$
  fields         $y_2, y_3$
  constraints    $J_{23}(y_2, y_3)$  .
```

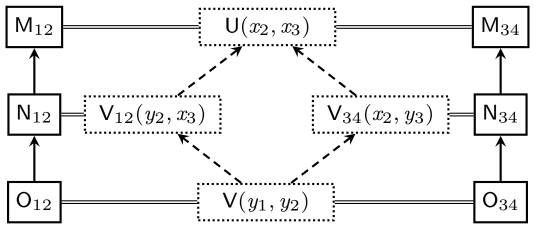

Figure 13: A lattice of interfaces

In developing M_{12}, x_2 is first instantiated by $h_2(y_2)$ (i.e., V_{12} instantiates U), subsequently x_3 is instantiated by $h_3(y_3)$ (i.e., V instantiates V_{12}). The development of M_{34} is similar, with an instantiation of x_3 followed by an instantiation of x_2. Note that this instantiation lattice satisfies the condition, mentioned earlier in Section 3, that the fields x_2 and x_3 are instantiated in distinct interfaces. Moreover, in the final interface V, the values given for x_2 and x_3 are consistent with the values used for instantiation in V_{12} and V_{34}.

At the same time, M_{12} and M_{34} are refined into O_{12} and O_{34}, relying on the interface instantiations. To be more precise, $\mathsf{M}_{12}(x_1, x_2, x_3)$[4] is first refined to $\mathsf{N}_{12}(y_1, y_2, x_3)$, relying on interface instantiation from U to V_{12}, and refinement of private variable x_1 to y_1. The refinement relationship between N_{12} and M_{12} is captured by the gluing invariant $J_{123}(x_1, x_2, y_1, y_2, x_3)$. In N_{12}, internal event e_1 and e_2 are refined by f_1 and f_2, respectively. Furthermore, external event a_3 is refined *equivalently* (see Section 3) to d_3.

Subsequently, $\mathsf{N}_{12}(y_1, y_2, x_3)$ is refined to $\mathsf{O}_{12}(y_1, y_2, y_3)$, relying on interface instantiation from V_{12} to V. In particular, in O_{12} internal events f_1 and f_2 are refined by g_1 and g_2, respectively. Furthermore, external event d_3 is refined

[4]We use the notation $\mathsf{M}(v)$ to denote that v are the variables of machine M.

equivalently to c_3. Similarly, machine $\mathsf{N}_{34}(x_2, y_3, y_4)$ refines $\mathsf{M}_{34}(x_2, x_3, x_4)$ with the gluing invariant $J_{234}(x_3, x_4, x_2, y_3, y_4)$. In particular, f_3 and f_4 are the refinement of internal events e_3 and e_4, respectively, and d_2 is the refinement of external event a_2. Subsequently, $\mathsf{N}_{34}(x_2, y_3, y_4)$ is refined by $\mathsf{O}(y_2, y_3, y_4)$, with internal events f_3 and f_4 refined by g_3 and g_4, and external event d_2 refined by c_2. The refinement relationships between the events are depicted in Fig. 14.

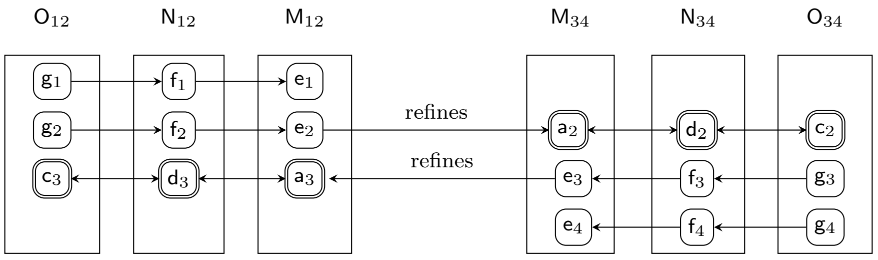

Figure 14: Maintaining the external invariant of several sub-models

5.2.2. *The output model*

Our output model is the composition machine O of O_{12} and O_{34} consists of internal events g_1, g_2 from O_{12}, and g_3 and g_4 from O_{34}. The correctness of our technique is guaranteed by proving that the composition O is a refinement of the original model M. In particular, the gluing invariants between O and M are the conjunctions of the gluing invariants that are used for refining the sub-machines M_{12} and M_{34}, i.e., J_{123}, J_{234}, together with the interface instantiation, i.e., $x_2 = h_2(y_2)$ and $x_3 = h_3(y_3)$.

In the subsequent, we focus on extracting assumptions and proving that g_2 is a refinement of e_2. Proofs related to other events are similar and omitted here.

5.2.3. *Extracting assumptions*

The assumption in proving the correctness of the output model is that the input model has been fully proved, i.e., the initial machine M, the interface instantiation and refinements N_{12}, N_{34}, O_{12}, O_{34}.[5] We focus on proving that g_2 is a refinement of e_2. Event e_2 is first refined into f_2 in N_{12} and subsequently into g_2 in O_{12} as follows:

$$\begin{array}{lcl}
\mathsf{event\ f_2\ refines\ e_2} & \widehat{=} & \mathsf{any}\ v_2\ \mathsf{where}\ F_2(v_2, y_1, y_2, x_3)\ \mathsf{then} \\
 & & \quad y_1, y_2, x_3 :\mid Q_2(v_2, y_1, y_2, x_3, y_1', y_2', x_3') \\
 & & \mathsf{end}\ , \\
\mathsf{event\ g_2\ refines\ f_2} & \widehat{=} & \mathsf{any}\ w_2\ \mathsf{where}\ G_2(w_2, y_1, y_2, y_3)\ \mathsf{then} \\
 & & \quad y_1, y_2, y_3 :\mid R_2(w_2, y_1, y_2, y_3, y_1', y_2', y_3') \\
 & & \mathsf{end}\ .
\end{array}$$

[5]The development is available at http://deploy-eprints.ecs.soton.ac.uk/364/

The fact that f_2 is a refinement of e_2 with invariant $J_{123} \land x_2 = h_2(y_2)$ and g_2 is a refinement of f_2 with invariant $x_3 = h_3(y_3)$ is captured by the following conditions.

$$\begin{array}{ll} & I_{123}(x_1, x_2, x_3) \land I_{23}(x_2, x_3) \land J_{123}(x_1, x_2, y_1, y_2, x_3) \land \\ & F_2(v_2, y_1, y_2, x_3) \land Q_2(v_2, y_1, y_2, x_3, y_1', y_2', x_3') \\ \Rightarrow & (\exists u_2, x_1', x_2' \cdot \\ & \quad E_2(u_2, x_1, x_2, x_3) \;\land\; P_2(u_2, x_1, x_2, x_3, x_1', x_2', x_3') \land \\ & \quad J_{123}(x_1', x_2', y_1', y_2', x_3') \;\land\; x_2' = h_2(y_2')) \;, \end{array} \tag{FE2}$$

$$\begin{array}{ll} & I_{123}(x_1, x_2, x_3) \land I_{23}(x_2, x_3) \land J_{123}(x_1, x_2, y_1, y_2, x_3) \land \\ & x_3 = h_3(y_3) \land \\ & G_2(w_2, y_1, y_2, y_3) \land R_2(w_2, y_1, y_2, y_3, y_1', y_2', y_3') \\ \Rightarrow & (\exists v_2, x_3' \cdot \\ & \quad F_2(v_2, y_1, y_2, x_3) \;\land\; Q_2(v_2, y_1, y_2, x_3, y_1', y_2', x_3') \land \\ & \quad x_3' = h_3(y_3')) \;. \end{array} \tag{GF2}$$

On the other hand, the key aspect for the correctness of our approach is that external events are refined equivalently. To be more precise, e_2 is projected as a_2 in M_{34} and is subsequently refined *equivalently* into d_2 in N_{34}.

$$\begin{array}{l} \textsf{event } \mathsf{a}_2 \;\widehat{=}\; \textsf{any } u_2, x_1 \textsf{ where } E_2(u_2, x_1, x_2, x_3) \textsf{ then} \\ \qquad\qquad\qquad x_2, x_3 :\mid \exists x_1' \cdot P_2(u_2, x_1, x_2, x_3, x_1', x_2', x_3') \\ \qquad\qquad\quad \textsf{end} \;, \\ \textsf{event } \mathsf{d}_2 \textsf{ refines } \mathsf{a}_2 \;\widehat{=}\; \textsf{any } t_2 \textsf{ where } D_2(t_2, x_2, y_3, y_4) \textsf{ then} \\ \qquad\qquad\qquad\qquad x_2, y_3 :\mid O_2(t_2, x_2, y_3, y_4, x_2', y_3') \\ \qquad\qquad\qquad\quad \textsf{end} \;. \end{array}$$

The fact that a_2 is a refinement of d_2 (“equivalently” refined) is captured by the following condition

$$\begin{array}{ll} & I_{23}(x_2, x_3) \land I_{234}(x_2, x_3, x_4) \land J_{234}(x_3, x_4, x_2, y_3, y_4) \land \\ & x_3 = h_3(y_3) \land \\ & E_2(u_2, x_1, x_2, x_3) \land (\exists x_1' \cdot P_2(u_2, x_1, x_2, x_3, x_1', x_2', x_3')) \\ \Rightarrow & (\exists t_2, y_3' \cdot \\ & \quad D_2(t_2, x_2, y_3, y_4) \;\land\; O_2(t_2, x_2, y_3, y_4, x_2', y_3') \land \\ & \quad J_{234}(x_3', x_4, x_2', y_3', y_4) \;\land\; x_3' = h_3(y_3')) \;. \end{array} \tag{AD2}$$

In particular, given that h_3 is a bijection, we have $x_3' = h_3(y_3')$ is equivalent to $y_3' = h_3^{-1}(x_3')$, which allows us to apply the one-point rule to simplify (AD2),

$$\begin{array}{ll} & I_{23}(x_2, x_3) \land I_{234}(x_2, x_3, x_4) \land J_{234}(x_3, x_4, x_2, y_3, y_4) \land \\ & x_3 = h_3(y_3) \land \\ & E_2(u_2, x_1, x_2, x_3) \land (\exists x_1' \cdot P_2(u_2, x_1, x_2, x_3, x_1', x_2', x_3')) \\ \Rightarrow & J_{234}(x_3', x_4, x_2', h_3^{-1}(x_3'), y_4) \;. \end{array} \tag{AD2-SIMP}$$

5.2.4. Proving consistency

We focus on proving that g_2 is a refinement of e_2 using the gluing invariants J_{123}, J_{234} and $x_2 = h_2(y_2)$ and $x_3 = h_3(y_3)$, i.e.,

$$\begin{array}{ll} & I_{123}(x_1, x_2, x_3) \wedge I_{23}(x_2, x_3) \wedge I_{234}(x_2, x_3, x_4) \wedge \\ & J_{123}(x_1, x_2, y_1, y_2, x_3) \wedge J_{234}(x_3, x_4, x_2, y_3, y_4) \wedge \\ & x_2 = h_2(y_2) \wedge x_3 = h_3(y_3) \wedge \\ & G_2(w_2, y_1, y_2, y_3) \wedge R_2(w_2, y_1, y_2, y_3, y_1', y_2', y_3') \\ \Rightarrow & (\exists u_2, x_1', x_2', x_3' \cdot \\ & \quad E_2(u_2, x_1, x_2, x_3) \ \wedge \ P_2(u_2, x_1, x_2, x_3, x_1', x_2', x_3') \wedge \\ & \quad J_{123}(x_1', x_2', y_1', y_2', x_3') \ \wedge \ J_{234}(x_3', x_4, x_2', y_3', y_4) \wedge \\ & \quad x_2' = h_2(y_2') \ \wedge \ x_3' = h_3(y_3')) \ . \end{array} \tag{GE2}$$

Assuming I_{123}, I_{23}, I_{234}, J_{123}, J_{234}, $x_2 = h_2(y_2)$, and $x_3 = h_3(y_3)$, we prove:

$$\begin{array}{lr} & G_2(w_2, y_1, y_2, y_3) \wedge R_2(w_2, y_1, y_2, y_3, y_1', y_2', y_3') \\ \Rightarrow & \text{(GF2)} \\ & (\exists v_2, x_3' \cdot F_2(v_2, y_1, y_2, x_3) \ \wedge Q_2(v_2, y_1, y_2, x_3, y_1', y_2', x_3') \ \wedge x_3' = h_3(y_3')) \\ \Rightarrow & \text{(FE2)} \\ & (\exists u_2, x_1', x_2', x_3' \cdot E_2(u_2, x_1, x_2, x_3) \wedge P_2(u_2, x_1, x_2, x_3, x_1', x_2', x_3') \wedge \\ & \quad J_{123}(x_1', x_2', y_1', y_2', x_3') \ \wedge x_2' = h_2(y_2') \wedge x_3' = h_3(y_3')) \\ \Rightarrow & \text{(AD2-SIMP)} \\ & (\exists u_2, x_1', x_2', x_3' \cdot E_2(u_2, x_1, x_2, x_3) \wedge P_2(u_2, x_1, x_2, x_3, x_1', x_2', x_3') \wedge \\ & \quad J_{123}(x_1', x_2', y_1', y_2', x_3') \ \wedge x_2' = h_2(y_2') \wedge x_3' = h_3(y_3') \wedge \\ & \quad J_{234}(x_3', x_4, x_2', h_3^{-1}(x_3'), y_4)) \\ \Leftrightarrow & \text{Equality } x_3' = h_3(y_3') \text{ and } h_3 \text{ bijective} \\ & (\exists u_2, x_1', x_2', x_3' \cdot E_2(u_2, x_1, x_2, x_3) \wedge P_2(u_2, x_1, x_2, x_3, x_1', x_2', x_3') \wedge \\ & \quad J_{123}(x_1', x_2', y_1', y_2', x_3') \ \wedge x_2' = h_2(y_2') \wedge x_3' = h_3(y_3') \wedge \\ & \quad J_{234}(x_3', x_4, x_2', y_3', y_4)) \ . \end{array}$$

The proof obligations presented in Section 3 are derived from the condition that external events are refined equivalently, i.e., the concrete event is a refinement of the abstract event and vice versa. The newly proposed proof obligations, i.e., *witness feasibility*, *guard weakening*, and *co-simulation*, are to prove that the abstract event is a refinement of the concrete event. They correspond to the standard proof obligations, i.e., *witness feasibility*, *guard strengthening*, and *simulation* in Event-B switching the roles of abstract and concrete machine. In particular, by making use of the same witness for proving equivalence between the abstract and concrete events, the proof obligations for *invariance preservation* are the same in both refinement directions.

6. Conclusion

We propose in this paper the notion of interface and interface instantiation for shared-variable decomposition in Event-B. An interface is a collection of external variables and their properties which can be shared between different submodels after a decomposition. Interface instantiation combines instantiation of carrier sets and constants with functional refinement of external variables. The encapsulation of external variables using interface offers us some flexibility in structuring the development using complex refinement and decomposition. In particular, we provide a practical method for refining external variables which is currently quite cumbersome [1].

The novelty of our approach is in the refinement of external events: we define additional proof obligations to ensure that the external events are refined *equivalently*. By contrast, in [1] equivalence is achieved by syntactical means replacing occurrences of abstract variables v by concrete terms $h(w)$. The proof obligations of our approach are similar to the standard proof obligations, even using the same refinement witnesses for proving the equivalence. We have presented a general technique for proving correctness of Event-B extensions, and showed how this is used to demonstrate the soundness of our approach. We illustrated the method by an industrial case study modelling a cruise control system.

The simple schema of contexts and instantiations shown in Fig. 8 is essential for comprehensibility. When extending our approach as presented in this article one should make sure that the simplicity is preserved. Complex schemas of instantiations of decomposed models could easily become incomprehensible. A strong methodology is needed to master interface instantiation or any generalisation of it. Keeping track of the changes to interfaces becomes quickly challenging in multiply decomposed and refined models.

Finally, we are looking at extending the Rodin tool [3] to support the notion of interface and interface instantiation.

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

References

[1] J.-R. Abrial. Event-B: Structure and Laws. In RODIN Deliverable 3.2 (`rodin.cs.ncl.ac.uk/deliverables/D7.pdf`), 2005.

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

[3] J.-R. Abrial, M. J. Butler, S. Hallerstede, T. S. Hoang, F. Mehta, and L. Voisin. Rodin: an open toolset for modelling and reasoning in Event-B. *STTT*, 12(6):447–466, 2010.

[4] J.-R. Abrial and S. Hallerstede. Refinement, Decomposition, and Instantiation of Discrete Models: Application to Event-B. *Fundam. Inform*, 77(1-2):1–28, 2007.

[5] D. A. Basin, A. Fürst, T. S. Hoang, K. Miyazaki, and N. Sato. Abstract data types in event-B - an application of generic instantiation. *CoRR*, abs/1210.7283, 2012.

[6] N. Evans and M. J. Butler. A Proposal for Records in Event-B. In J. Misra, T. Nipkow, and E. Sekerinski, editors, *FM*, volume 4085 of *LNCS*, pages 221–235. Springer, 2006.

[7] R. Gmehlich, K. Grau, S. Hallerstede, M. Leuschel, F. Lösch, and D. Plagge. On fitting a formal method into practice. In S. Qin and Z. Qiu, editors, *ICFEM*, volume 6991 of *LNCS*, pages 195–210. Springer, 2011.

[8] S. Hallerstede. The Event-B Proof Obligation Generator, 2005.

[9] S. Hallerstede and T. S. Hoang. Refinement by interface instantiation. In J. Derrick, J. A. Fitzgerald, S. Gnesi, S. Khurshid, M. Leuschel, S. Reeves, and E. Riccobene, editors, *ABZ*, volume 7316 of *LNCS*, pages 223–237. Springer, 2012.

[10] T. S. Hoang and J-R. Abrial. Event-B Decomposition for Parallel Programs. In *ABZ2010*, volume 5977 of *LNCS*, pages 319–333. Springer, 2010.

[11] A. Iliasov, E. Troubitsyna, L. Laibinis, A. Romanovsky, K. Varpaaniemi, D. Ilic, and T. Latvala. Supporting reuse in Event B development: Modularisation approach. In M. Frappier, U. Glässer, S. Khurshid, R. Laleau, and S. Reeves, editors, *ASM*, volume 5977 of *LNCS*, pages 174–188. Springer, 2010.

[12] M. Jackson. *Problem Frames: Analyzing and structuring software development problems.* Addison-Wesley Longman Publishing Co., Inc., 2001.

[13] L. Ladenberger, J. Bendisposto, and M. Leuschel. Visualising event-B models with B-motion studio. In M. Alpuente, B. Cook, and C. Joubert, editors, *FMICS*, volume 5825 of *LNCS*, pages 202–204. Springer, 2009.

[14] M. Leuschel and M. J. Butler. ProB: an automated analysis toolset for the B method. *STTT*, 10(2):185–203, 2008.

[15] F. Loesch, R. Gmehlich, K. Grau, C. B. Jones, and M. Mazzara. DEPLOY Deliverable D19: Pilot Deployment in the Automotive Sector.

[16] M. Poppleton. The composition of event-B models. In E. Börger, M. J. Butler, J. P. Bowen, and P. Boca, editors, *ABZ*, volume 5238 of *LNCS*, pages 209–222. Springer, 2008.

[17] S. Schneider and H. Treharne. Changing system interfaces consistently: A new refinement strategy for $CSP||B$. *Sci. Comput. Program.*, 76(10):837–860, 2011.

[18] R. Silva, C. Pascal, T. S. Hoang, and M. Butler. Decomposition tool for event-B. *Softw, Pract. Exper*, 41(2):199–208, 2011.

Appendix A. Summary of Event-B Symbols

Symbols	Meaning
$S \setminus T$	Set difference
$r[S]$	Relational image
$(\lambda x \cdot P \mid E)$	Lambda expression
$x \mapsto y$	Ordered pair
r^{-1}	Inverse relation
$r \mathbin{;} s$	Forward composition
$m \mathinner{..} n$	Integer range
$\mathbb{P}(S)$	Power set
$S \times T$	Cartesian product
$x :\mid P$	Non-deterministic assignment (becomes such that)
$x :\in S$	Non-deterministic assignment (becomes in set)
$x := E$	Deterministic assignment

Table A.3: Summary of Event-B Symbols