Supporting Reuse Mechanisms for Developments in Event-B: Composition

Renato Silva and Michael Butler

School of Electronics and Computer Science
University of Southampton, UK
{ras07r,mjb}@ecs.soton.ac.uk

Abstract. The development of specifications often is a combination of smaller sub-components. Focusing on reuse, an interesting perspective is to formally define the combination of sub-components through refinement steps, reusing their properties and generating larger systems. The previous situation suggests the application of a reuse mechanism: *composition*. Event-B is a formal method that allows modelling and refinement of systems. The combination and reuse of existing sub-components is not currently supported in Event-B. We propose the development of composition by extending the Event-B formalism as an option for developing larger models, focusing in distributed systems. A tool is developed to support the shared event composition in the Rodin platform. Properties and proof obligations of sub-components are reused and sufficient proof obligations are generated to ensure valid composed models.

Key words: formal methods, composition, Event-B, specification, design techniques

1 Introduction

The development of specifications in a "top-down" style starts with an abstract model of the envisaged system. Often that system is a combination of several sub-components. Instead of creating a large system from scratch, interacting sub-components can be composed. In a computer science context, (functional) composition can be defined as the act or mechanism to combine simple functions to build more complicated ones. It derives from an usual mathematical step where the composition of functions results in each function to be passed as the argument of the next; the result of the last one is the result of the whole. In the formal methods context, in particular for specifications, *composition* is the capacity to model the interaction of sub-components generating larger and more concrete specifications. Usually the interaction between systems is based on shared state, shared operations/events [1] or a combination of both (for example fusion composition [2,3]). We follow the shared event approach for composition inspired by CSP [4]. We extend Event-B to support shared event composition including the definition of static checks and proof obligations for a composed machine and outlining a tool extension in the Rodin platform [5], the toolset for Event-B [6,7].

This document is structured as follows: Sect. 2 gives an overview of the Event-B formal method. Section 3 introduces the notion and motivation for shared event composition. The notion of composed machine and respective static checks and proof obligations are introduced using a simple case study in Sect. 4. Section 5 describes the tool developed to support the shared event composition and Sect. 6 illustrates the application of composition as a reuse mechanism to a more complex case study: a railway system. Related work, conclusions and future work are drawn in Sect. 7.

2 Event-B Language

Event-B is a formal methodology that uses mathematical techniques based on set theory and first order logic allowing the specification of systems. An abstract Event-B specification is divided into two parts: a static part called *context* and a dynamic part called *machine*. A machine *SEES* contexts that consist of carrier sets (types), constants and axioms of the system. A machine contains the state variables whose values are assigned in *events*. Events can only occur when enabled by their *guards* being true and as a result *actions* are executed. Events can have *parameters* that are local variables to the event and can be used by the guards or by the actions. An event *evt* with a set of parameters *t*, a set of guards *G* and a set of actions *S* assigning variable *v* has the following shape:

$$evt \mathrel{\widehat{=}} \textbf{ANY}\ t\ \textbf{WHERE}\ G(t,v)\ \textbf{THEN}\ S(t,v)\ \textbf{END}$$

For an assignment of variable v, before-after predicate denotes the relationship holding between the state variables of the model just before (denoted by v) and just after (denoted by $v\prime$) applying a substitution, which can be deterministic ($v\prime := E(v)$) or non-deterministic ($v\prime :\in E(v)$ or $v\prime :\mid E(v)$), where $E(v)$ denotes an expression using v. The *INVARIANT* defines the dynamic properties of the specification. Proof obligations (POs) are generated to verify that the invariant is maintained before and after an event is enabled (invariant preservation PO). POs are an important part of the definition of Event-B. Later we will extend the standard POs to cover composition.

An abstract Event-B specification can be refined by adding more details and becoming more concrete. Refinement of a machine consists in refining existing events. The relation between variables in the concrete and abstract model is given by a *gluing invariant*. Proof obligations are generated to ensure that this invariant is preserved in the concrete model. New events can be introduced in a refinement of an abstract model. Those events are not visible to the environment as they represent internal events (hidden) that must refine *skip*. New events must not prevent abstract events to be executed (divergence). This is proved if each new event decreases a *variant* [8] (in first approximation, could be a natural number but it can be lexicographic). Next we give more details on shared event approach.

3 Shared Event Approach

The shared event approach is suitable for the development of distributed systems [9]: sub-components interact through synchronised events in parallel; moreover sub-components can communicate using shared parameters which is useful for modelling message passing systems. Shared event composition is achieved through events interaction between sub-components. Although the interacting systems have states, the communication occurs at the event level. CSP models parallel processing and interaction between systems [10]. Due to the stateless CSP approach, several works try to combine state-based and event-based approaches, as are the examples of combining CSP and B [11,12,13] or combining CSP with object oriented classes [14,15]. Butler [16] proposes a shared event (de)composition for Event-B inspired by CSP and action systems with event sharing as seen in Fig. 1. This kind of (de)composition partitions the variables into sub-components and variable sharing is not permitted. We follow that work in our approach.

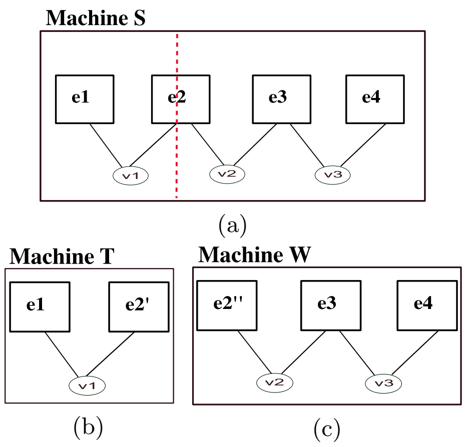


Fig. 1. Shared Event Decomposition of Machine *S* in Machines *T* and *W* with shared event *e2*

In Fig. 1, machine *S* has events *e1* to *e4* and variables *v1* to *v3*. That machine is to be decomposed so that *v1* is placed in sub-component *T* and *v2* and *v3* are placed in *W*. Events using variables allocated to different sub-components (event *e2* shares *v1* and *v2*) must be split. The part corresponding to each variable (*e2'* and *e2"*) is used to create partial versions of the non-decomposed event.

To see how *e2* is decomposed into *e2'* and *e2"*, we first consider the composition of events. The composition of synchronised events generates a new event whose guard is the conjunction of the original guards and the actions are executed in parallel. An interesting situation occurs when events to be composed have the same parameter name.

Definition 1 *If both events $evt1$ and $evt2$ have a parameter t:*

$$evt1 \mathrel{\hat{=}} \boldsymbol{ANY}\ t?, x\ \boldsymbol{WHERE}\ t? \in A \wedge G(t?, x, m)\ \boldsymbol{THEN}\ S(t?, x, m)\ \boldsymbol{END}$$
$$evt2 \mathrel{\hat{=}} \boldsymbol{ANY}\ t!, y\ \boldsymbol{WHERE}\ H(t!, y, n)\ \boldsymbol{THEN}\ T(t!, y, n)\ \boldsymbol{END}$$

then [9]:

$$evt1 \parallel evt2 \mathrel{\hat{=}}$$
$$\boldsymbol{ANY}\ t!, x, y\ \boldsymbol{WHERE}\ t! \in A \wedge G(t!, x, m)\ \wedge\ H(t!, y, n)$$
$$\boldsymbol{THEN}\ S(t!, x, m) \parallel T(t!, y, n)\ \boldsymbol{END}$$

where x, y, t are sets of parameters from each of the events $evt1$ and $evt2$. Event $evt1$ has $t?$ as an input parameter and $evt2$ has $t!$ as an output parameter and the resulting composition is $t!$ itself an output parameter (like in CSP). This property can be used to model message broadcasting systems: $evt2$ sends a message to $evt1$ using the parameter t. The types of the shared parameter must match or be a subset of each other:

$$t! \in A \wedge t? \in B \Rightarrow A \cap B \neq \varnothing$$

where A and B are types (carrier sets). Events with shared parameters of type input can also be composed and the shared parameter has an input behaviour. Events with shared parameters of type output cannot be composed since this could lead to a deadlock state [9].

Event-B has the same semantics structure and refinement definitions as action systems. It is possible to make a correspondence between parallel composition in CSP and an event-based view of parallel composition for action systems [1,17].

Definition 2 *A failure-divergence definition (CSP) can be applied to Event-B machines:*

- *$S \in Machine \rightarrow FD$, where FD is the set of Failure-Divergence for Machine*
- *$PAR(P, Q)$ where $P, Q \in FD$ and PAR defines the semantics of the process $P \parallel Q$ in CSP*
- *then $S(M \parallel N) = PAR(S(M), S(N))$*

The semantics of the parallel composition of machines M and N corresponds to the set of failure-divergence for each individual machine in parallel. Since PAR is monotonic machines M and N can be refined independently and that is one of the most important and powerful properties that shared event composition in Event-B inherited from CSP. After the (de)composition, the individual machines can be further refined since the composition relation holds.

3.1 Composition

Composition has been object of study for a long time. The possibility of executing several actions at once in *parallel* is very attractive since it minimises the effort of sequential execution and it is expected to accelerate the execution process. We focus on shared event parallel composition of specifications applied to Event-B using a simple case study: *Simple Composition*. Suppose we have machine *m1* containing a variable *x*. After the initialisation, event *dec* decrements *x* non-deterministically. We also have a machine *m2* with a variable *y* that is incremented in the event *inc* as seen in Fig. 2.

```
machine m1

variables x

invariants
  @inv1 x ∈ ℕ

events
  event INITIALISATION
    then
      @act1 x ≔ 100
  end

  event dec
    any i
    where
      @grd2 x > 0
      @grd1 i ∈ 1‥x
    then
      @act1 x ≔ x−i
  end
end
```

(a)

```
machine m2

variables y

invariants
  @inv1 y ∈ ℕ

events
  event INITIALISATION
    then
      @act1 y ≔ 0
  end

  event inc
    any i
    where
      @grd1 i ∈ ℕ
    then
      @act1 y ≔ y+i
  end
end
```

(b)

```
machine cm

variables x y

invariants
  @inv1 x ∈ ℕ
  @inv2 y ∈ ℕ
  @inv3 x+y = 100

events
  event INITIALISATION
    then
      @act1 x ≔ 100
      @act2 y ≔ 0
  end
  event transfer
    any i
    where
      @grd1 x > 0
      @grd2 i ∈ 1‥x
    then
      @act1 x ≔ x−i
      @act2 y ≔ y+i
  end
end
```

(c)

Fig. 2. Machines *m1*, *m2* and *cm* resulting from the composition of *m1* and *m2*

We compose these two machines using shared event composition. The outcome of this composition is a new machine where events *INITIALISATION* of machines *m1* and *m2* are composed in parallel, the variables of each of the machines are merged and the invariants are conjoined. In addition,we add an invariant combining both variables: *x + y =100*. Event *dec* from machine *m1* (represented as *m1.dec*) and the event *inc* from machine *m2* (*m2.inc*) are composed using Def. 1:

$$m1.dec \mathrel{\widehat{=}} \textbf{ANY } i! \textbf{ WHERE } x > 0 \land i \in 1 \mathinner{..} x \textbf{ THEN } x := x - i \textbf{ END}$$
$$m2.inc \mathrel{\widehat{=}} \textbf{ANY } i? \textbf{ WHERE } i \in \mathbb{N} \textbf{ THEN } y := y + i \textbf{ END}$$

$$m1.dec \parallel m2.inc \mathrel{\widehat{=}}$$
$$\textbf{ANY } i! \textbf{ WHERE } x > 0 \land i \in 1..x \land i \in \mathbb{N} \textbf{ THEN } x := x-i \parallel y := y+i \textbf{ END}$$

Figure 2(c) shows the machine resulting from composing *m1* and *m2* where *m1.dec* ∥ *m2.inc* is named *transfer*. Although it is interesting to have achieved the composition of individual machines, in general we construct models using a "top-down" approach starting from a simple, abstract model and refining it to become more concrete and closer to an implementation [18]. Therefore a more interesting option is to use the composed machine as a refinement of an abstract model. Having ensured the refinement link between abstract and concrete model (usually by discharging refinement POs), one could use the monotonicity property to refine the initial sub-components without "breaking" the composition link, which is one of the greatest advantages of this kind of composition. Based on this simple example, next section introduces the notion of composed machine as a way to reuse existing sub-components to build larger components.

4 Composed Machines

Instead of having to manually compose *m1* and *m2* in Figs. 2(a) and (b) to get *cm* of Fig. 2(c), we introduce a new construct, a *composed machine* representing the shared event composition. Next we describe the structure of a composed machine, the required static checks and proof obligations to validate the composition.

4.1 Structure of Composed Machines

We shall describe the structure of a composed machine using the *Simple Composition* example depicted in Fig. 2. The corresponding composed machine *cm* can be seen in Fig. 3.

```
COMPOSED MACHINE cm
REFINES cm0
INCLUDES
      m1
      m2
INVARIANTS
      x + y = 100
EVENTS
      transfer refines cm1.transfer
            Combines Events m1.dec || m2.inc
END
```

Fig. 3. Composed Machine cm2

Composed machine *cm* refines an abstract machine *cm0* and includes machines *m1* and *m2*. An invariant "connecting" the two machined is added. Event *cm0.transfer* is refined as the parallel composition (interaction) of *m1.dec* and *m2.inc*. The outcome of this composition is similar to Fig. 2(c).

Figure 4 represents some standard machines that will be used to outline a generic structure for a composed machine. Machines *M1* to *Mm* are composed and the resulting composed machine *CM1* (Fig. 5) is a refinement of an abstract machine *CM0*. Contexts seen by the included machines are implicitly seen by the

```
MACHINE M1
VARIABLES ...
INVARIANTS ...
EVENTS
    INITIALISATION
    $evt_{11}$
    ...
    $evt_{1p}$
END
```
(a)

```
MACHINE M2
VARIABLES ...
INVARIANTS ...
EVENTS
    INITIALISATION
    $evt_{21}$
    ...
    $evt_{2q}$
END
```
(b)

...

```
MACHINE Mm
VARIABLES ...
INVARIANTS ...
EVENTS
    INITIALISATION
    $evt_{m1}$
    ...
    $evt_{mr}$
END
```
(c)

Fig. 4. Generic machines *M1* to *Mm*

composed machine. Additional contexts can be added using the *SEES* section and gluing invariants can be added in the *INVARIANT* section. Afterwards, the generated machine *CM1* can be further refined.

```
COMPOSED MACHINE CM1
REFINES CM0
INCLUDES
    M1
    ...
    Mm
SEES -                          /*Additional contexts*/
INVARIANT -                     /*further invariants and/or gluing invariant*/
EVENTS
    $evt_{01}$ refines $CM0.evt_{01}$
        Combines Events $M1.evt_{11}$ || $M2.evt_{21}$ ... || $Mm.evt_{m1}$
    $evt_{02}$ refines $CM0.evt_{02}$
        Combines Events $M1.evt_{12}$ || $M2.evt_{22}$ ... || $Mm.evt_{m2}$
    ...
    $evt_{0n}$ refines $CM0.evt_{0n}$
        Combines Events $M1.evt_{mp}$ || $M2.evt_{2q}$ ... || $Mm.evt_{mr}$
END
```

Fig. 5. A generic Composed Machine

Next we present more details about the composed machine by introducing the static checks that are required to validate composed machines.

4.2 Static Checks

Composed machines need to be validated against some well-formedness conditions. Some of those conditions are defined as follows:

– Variables of included machines cannot be shared.
– In a refinement, the composed events must refine all abstract events.
– A composed machine is defined by including at least one machine.
– Gluing invariants use variables, sets and constants from included machines.
– A composed event is defined by at least one single event.
– When several events are combined into a composed event, those events must be from different included machines.

Next we present the POs that need to be generated for composed machines.

4.3 Proof Obligations

POs play an important role in Event-B developments. We use context Ctx, machines CM_0, M_1, M_2 and composed machine CM_1 in Fig. 6 to express the POs that need to be generated for composition.

```
CONTEXT $Ctx$
CONSTANTS $c$
SETS $s$
AXIOMS $A(s,c)$
```
(a)

```
MACHINE $CM_0$ SEES $Ctx$
VARIABLES $v$
INVARIANT $I(s,c,v)$
EVENT $evt \widehat{=}$
    ANY x WHERE
        $G(x,s,c,v)$
    THEN
        $v :\mid BAP(x,s,c,v,v')$
    END
```
(b)

```
MACHINE $M_1$ SEES $Ctx$
VARIABLES $v_1$
INVARIANT $I_1(s,c,v_1)$
EVENT $evt1 \widehat{=}$
    ANY $x_1$ WHERE
        $G_1(x_1,s,c,v_1)$
    THEN
        $v_1 :\mid BAP_1(x_1,s,c,v_1,v_1')$
    END
```
(c)

```
MACHINE $M_2$ SEES $Ctx$
VARIABLES $v_2$
INVARIANT $I_2(s,c,v_2)$
EVENT $evt2 \widehat{=}$
    ANY $x_2$ WHERE
        $G_2(x_2,s,c,v_2)$
    THEN
        $v_2 :\mid BAP_2(x_2,s,c,v_2,v_2')$
    END
```
(d)

```
COMPOSED MACHINE $CM_1$
REFINES $CM_0$
INCLUDES
    $M_1$
    $M_2$
INVARIANT $J(s,c,v_1,v_2)$
EVENTS
    evt refines $CM_0$.evt
        Combines Events $M1.evt1 \parallel M2.evt2$
    END
```
(e)

Fig. 6. Context Ctx, abstract machine CM_0, included machines M_1 and M_2 and composed machine CM_1

Context Ctx is characterised by the constants c, the carrier sets s and the axioms *A(s,c)*. This context is seen by all the involved machines. The abstract machine CM_0 contains a set of variables v, a list of invariants *I(s,c,v)* and an event *evt* defined by the parameter *x*, guard *G(x,s,c,v)* and before-after predicate $BAP(x,s,c,v,v\prime)$ over the set of variables v. Similarly machines M_1 and M_2 are defined with their respective events. Finally composed machine CM_1 refines CM_0, includes machines *M1* and *M2*, contains a set of additional invariants $J(s,c,v_1,v_2)$ and composes event *evt*. For simplicity we define the POs in terms of a composition of 2 machines. The rules generalise easily to composition of n machines. A proof obligation is a sequent of the shape:

$$\boxed{\begin{array}{l} \textit{Hypothesis} \\ \vdash \textit{Goal} \end{array}}$$

[19] defines a list of standard POs for contexts and machines. We extend some of those POs, in particular the ones related with refinement: invariant preservation (INV), guard strengthening (GRD), simulation (SIM) and well-definedness (WD) to composed machines using CM_1. For instance, invariant preservation PO of an invariant "inv" in event "evt1" in a standard machine

like *M1* in Fig. 6(c) is given by (1). $i(s,c,v_1')$ is one of the invariants in the set $I_1(s,c,v_1)$ where v_1 is replaced by the after-state v_1'.

$$\boxed{\text{evt1/inv/INV: } \begin{array}{l} A(s,c) \\ I_1(s,c,v_1) \\ G_1(x_1,s,c,v_1) \\ BAP_1(x_1,s,c,v_1,v_1') \\ \vdash i(s,c,v_1') \end{array}} \tag{1}$$

Following the previous, the POs that need to be generated for a composition machine CM_1 are:

Invariant Preservation (INV): The invariant of the composed machine is the conjunction of the invariants of each included machine. We assume that the invariant preservation proof obligation in the included machines are already discharged so we just need to deal with the gluing invariants that are added to the composed machine $[J(s,c,v_1,v_2)]$. It is just necessary to discharge the POs related with the additional gluing invariant for each composed event. In Fig. 6(e), for event "evt" and one of the invariants "inv" in $J(s,c,v_1,v_2)$ the "INV" PO is represented by (2).

$$\boxed{\text{evt/inv/INV: } \begin{array}{l} A(s,c) \\ J(s,c,v_1,v_2) \wedge I_1(s,c,v_1) \wedge I_2(s,c,v_2) \\ G_1(x_1,s,c,v_1) \wedge G_2(x_2,s,c,v_2) \\ BAP_1(x_1,s,c,v_1,v_1') \wedge BAP_2(x_2,s,c,v_2,v_2') \\ \vdash j(s,c,v_1',v_2') \end{array}} \tag{2}$$

Guard Strengthening (GRD): In a refinement, it is necessary to ensure that when concrete events are enabled then the corresponding abstract ones are also enabled. For event "evt" of Fig. 6(e) and abstract guard "grd" in the corresponding abstract event in Fig. 6(b), the "GRD" PO is represented by (3).

$$\boxed{\text{evt/grd/GRD: } \begin{array}{l} A(s,c) \\ I(s,c,v) \\ J(s,c,v_1,v_2) \wedge I_1(s,c,v_1) \wedge I_2(s,c,v_2) \\ G_1(x_1,s,c,v_1) \wedge G_2(x_2,s,c,v_2) \\ \vdash G(x,s,c,v) \end{array}} \tag{3}$$

Simulation (SIM): Ensures that each action in a concrete composed event simulates the corresponding abstract action. The goal to be proved requires that exists an abstract action (BA) in the abstract before-after predicate (BAP) such that when a concrete event is executed, the corresponding abstract event is not contradicted. In Fig. 6(e), for event "evt" and action "act" in both abstract and concrete event, the simulation proof obligation "SIM" is represented by (4).

$$\boxed{\text{evt/act/SIM:}\quad \begin{array}{l} A(s,c) \\ I(s,c,v) \\ J(s,c,v_1,v_2) \wedge I_1(s,c,v_1) \wedge I_2(s,c,v_2) \\ G_1(x_1,s,c,v_1) \wedge G_2(x_2,s,c,v_2) \\ BAP_1(x_1,s,c,v_1,v_1') \wedge BAP_2(x_2,s,c,v_2,v_2') \\ \vdash \exists BA \cdot BAP(x,s,c,v,v') \end{array}} \qquad (4)$$

Well-Definedness(WD): Ensures that a potential invariant(inv) is indeed well-defined. This proof obligation is only generated for the added gluing invariants in the composed machine as we assume the invariants in the included machines are already discharged and can be reused.

Feasibility POs (FIS) ensure that each non-deterministic action is feasible. Composed machines use the actions of the composed events, so the respective POs generated in the included machine are reused. Besides the above POs, we would like to add the enabledness proof obligation thus ensuring that an event disabled in the concrete model is also disabled in the abstract model. Currently Rodin platform does not support these kind of POs. Enabledness POs are not only applicable for composition and it is a subject that has been studied already. Further study is required to support this kind of the enabledness properties in the Rodin platform before we can apply it to the shared event composition. It is possible to reuse machines POs because these are the POs that would result from expanding CM_1 as explained in Sect. 3. Note that to discharging POs, the order in which the hypotheses are presented is relevant. That is why we can reuse some of the POs that are already discharged in the included machine.

Next we present the tool that was developed as a plug-in in the Rodin platform to support the shared event composition for Event-B.

5 Shared Event Composition Plug-in

The Rodin platform is an extensible open source tool, based on Eclipse [20] where it is possible to add components/funtionalities using "add-on" plug-ins. Rodin supports a static checker that validates system properties (lexical analyser, syntactic analyser and type checker) and automatically generates proof obligations for machines and contexts. A plug-in for composed machines was developed to support the shared event composition. We extend the Rodin static checker to validate composed machine based on checks defined in Sec. 4.2. POs should be automatically generated over the composed machines similarly to what happens to machines and contexts. Currently this is not done but we will address this issue in the future. The current solution to address POs is to generate a standard machine from the composed machine properties (expand) as seen in Fig. 7(a) (machine $M2'$) and then use the existing proof obligation generation.

In the future, we would like to still have the option to generate a new machine but the POs should be discharged at the composed machine as depicted in Fig.

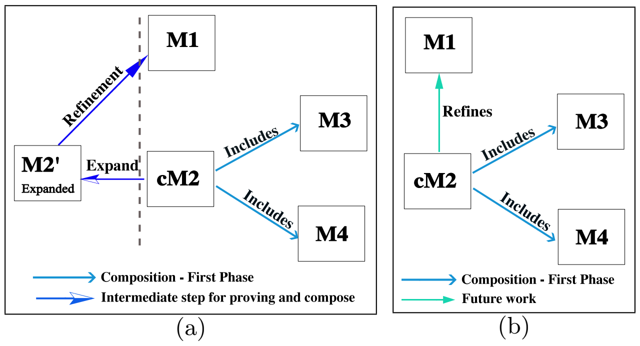


Fig. 7. Composition structure: current and future

7(b). Generating a new machine allows the further development of the composed model. Moreover it allows the visualisation of the composed events which it is beneficial based on the experience of using the tool. Next section presents the application of the shared event composition and tool to a more complex case study whose architecture is a typical distributed system.

6 RailWay System

The railway system case study describes a formal approach for the development of embedded controllers for a railway based on [21]. The railway system is characterised by trains, tracks (also called sections or cdv) and a communication entity between trains and tracks. The trains are in sections and whenever a train enters/leaves a section, a notification is sent to the track entity. In case of an potential hazard, the track entity orders the trains to brake. The track entity is responsible for controlling the sections, changing switch directions (switch are special tracks that can be divergent or convergent) and sending signalling messages to the trains through the communication entity. Figure 8[1] shows an schematic representation of the railway system decomposed into subsystems.

The abstract model of the railway system does not have notion of sub-components and the system is seen as a single component. After a refinement, the system is manually decomposed. Our goal is to be able to refine the sub-components independently. Nevertheless, we need to prove that the composition of the sub-components is a refinement of the abstract model. The shared event composition plug-in is used to compose the sub-components and by discharging the POs in the generated composed machine it is ensured that it is a refinement of the abstraction model. The composed machine corresponding to the composition of *Trains*, *Track* and *Comms* is depicted in Fig. 9. All *RailWay_M1* events are refined as composition of events from sub-components *Track_M0*, *Trains_M0* and *Comms_M0*.

[1] Image extracted from [21]

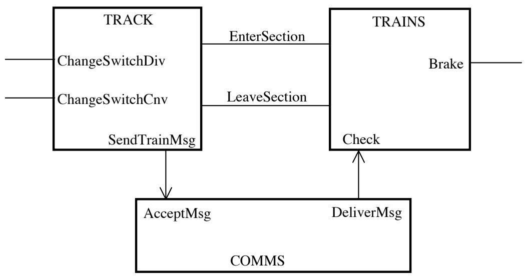


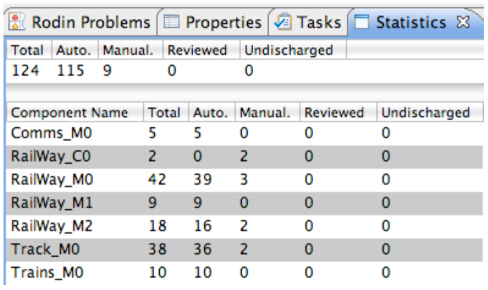
Fig. 8. Components of railway system

```
COMPOSED MACHINE Railway_M2
REFINES Railway_M1
INCLUDES
    Track_M0
    Trains_M0
    Comms_M0
EVENTS
    SendTrainMsg refines SendTrainMsg
        Combines Events Track_M0.SendTrainMsg || Comms_M0.Send
    RecvTrainMsg refines RecvTrainMsg
        Combines Events Trains_M0.RecvTrainMsg || Comms_M0.Recv
    ChangeSpeed refines RecvTrainMsg
        Combines Events Trains_M0.ChangeSpeed
    Brake refines Brake
        Combines Events Trains_M0.Brake
    EnterCDV refines enterCDV
        Combines Events Trains_M0.EnterCDV || Track_M0.EnterCDV
    LeaveCDV refines LeaveCDV
        Combines Events Trains_M0.LeaveCDV || Track_M0.LeaveCDV
    SwitchChangeDiv refines SwitchChangeDiv
        Combines Events Track_M0.SwitchChangeDiv
    SwitchChangeCnv refines SwitchChangeCnv
        Combines Events Track_M0.SwitchChangeCnv
END
```

Fig. 9. Composed machine *RailWay_M2* for the railway system

A summary of the POs for the development can be seen in Fig. 10. From the overall 124 generated POs, only 9 have to be interactively discharged. The most abstract model *RailWay_M0* has the majority of the POs resulting from the definition of the important events that constitute the railway system. *RailWay_M1* has only 9 POs due to the introduction of the communication layer. *RailWay_M2* is the expanded version of the composed machine and has mostly refinement POs due to the manipulation of the events in each sub-component.

The manual decomposition partition of the system into sub-components also partition the POs which are less for each sub-component and expected to be easier to discharge. The manual POs are mostly related with finiteness of sets and are easily discharged. Having ensured the validity of the composition, sub-components can be further refined which is one of the most important objectives of using composition in large system specifications.



Total	Auto.	Manual.	Reviewed	Undischarged
124	115	9	0	0

Component Name	Total	Auto.	Manual.	Reviewed	Undischarged
Comms_M0	5	5	0	0	0
RailWay_C0	2	0	2	0	0
RailWay_M0	42	39	3	0	0
RailWay_M1	9	9	0	0	0
RailWay_M2	18	16	2	0	0
Track_M0	38	36	2	0	0
Trains_M0	10	10	0	0	0

Fig. 10. Proof obligation statistics for the railway system development

7 Conclusions

Composition allows the interaction of components and usually occurs through variable sharing, event sharing or a combination of both like fusion composition [2,3]. Back [22], Abadi and Lamport [23] studied the interaction of components through shared variable composition. Jones [24] also proposes a shared variable composition for VDM by restricting the behaviour of the environment and the operation itself in order to consider the composition valid using rely-guarantee conditions. In classical B the composition [18,25,26] uses keywords like *Includes* to extend a machine, not allowing writing access to variables in the included machine or keyword *Sees* used to complement machines. Bellergarde et at [27] suggests a synchronised parallel composition of event systems in B where subsystems are specified as isolated. The interaction is achieved by synchronised events under feasibility conditions and the component composition is seen as a labelled transition system. The objective is to verify the refinement of synchronised parallel composition between components. This approach is limited to finite state components but under these conditions avoids all the variant requirements for design and proof. That work uses the B event system framework while our composition follows synchronisation and communication in the CSP style. [28] discusses a combination of action systems and classical B by composing machines using parallel systems in an action system style and preserving the invariants of the individual machines. This approach allows the classical B to derive parallel and distributed systems and since the parallel composition of action system is monotonic, the subsystems in a parallel composition may be refined independently. Still using classical B in [29] a global view of the system is refined by the manual (de)composition of sub-components. Event parameters are use for interaction between sub-components. However the refinement relation it is not easy to achieve because distributed systems are not a refinement machine in the B sense. We untangled this problem in Event-B showing that is possible to prove that refinement relation. Abrial et al [6,30] propose a state-based (de)composition for Event-B introducing the notion of shared variables and external events.

Our composition has an event-based behaviour inspired by CSP and action systems [1]. Shared event composition is monotonic and the used sub-components can be further refined independently. Thus refinement as a “top-down” style for creating specifications is allowed including the generation of the respective POs. The interaction of components with value-passing is possible through the use of event parameters. We extend Event-B to support shared event composition, allowing combination and reuse of existing sub-components through the introduction of *composed machines*. Required static checks are developed and POs are generated to validate the composition. Composed machines can be further refined independently from the refinement of sub-components. Such approach seems suitable for modelling distributed systems, where the system can be seen as a whole (composed component) or a combination of interacting parts (sub-components). A tool is developed to support composition in the Rodin platform. A railway case study defining a classic distributed system is modelled using the composition tool. With the developed work, we expect to have the necessary conditions to develop another reuse technique that can be seen as the inverse operation of composition: decomposition.

References

1. Butler, M.J.: A CSP Approach to Action Systems. PhD thesis, Oxford University (1992)
2. Back, R.J.R., Butler, M.J.: Fusion and Simultaneous Execution in the Refinement Calculus. Acta Informatica **35**(11) (1998) 921–949
3. Poppleton, M.: The composition of Event-B models. In: ABZ2008: Int. Conference on ASM, B and Z. Volume 5238., Springer LNCS (September 2008) 209–222
4. Hoare, C.A.R.: Communicating Sequential Processes. Prentice Hall International Series in Computer Science (1985)
5. Rodin: RODIN project Homepage. http://rodin.cs.ncl.ac.uk (September 2008)
6. Métayer, C., Abrial, J.R., Voisin, L.: Event-B Language. Technical report, Deliverable 3.2, EU Project IST-511599 - RODIN (May 2005)
7. Abrial, J.R., Butler, M.J., Hallerstede, S., Voisin, L.: An Open Extensible Tool Environment for Event-B. In: ICFEM. (2006) 588–605
8. Abrial, J.R., Cansell, D., Méry, D.: Refinement and Reachability in Event_B. ZB 2005: Formal Specification and Development in Z and B (2005) 222–241
9. Butler, M.: An Approach to the Design of Distributed Systems with B AMN. In: Proc. 10th Int. Conf. of Z Users: The Z Formal Specification Notation (ZUM), LNCS 1212. (1997) 221–241
10. Hoare, C.A.R.: Communicating sequential processes. Commun. ACM **21**(8) (1978) 666–677
11. Butler, M.J.: csp2B: A Practical Approach to Combining CSP and B. Formal Aspects of Computing **12** (2000) 182–196
12. Butler, M., Leuschel, M.: Combining CSP and B for Specification and Property Verification. In Fitzgerald, J., Hayes, I., Tarlecki, A., eds.: Formal Methods 2005. Number 3582 in LNCS, Springer (January 2005) 221–236
13. Treharne, H., Schneider, S.: Using a Process Algebra to Control B Operations. In: IFM ’99: Proceedings of the 1st International Conference on Integrated Formal Methods, London, UK, Springer-Verlag (1999) 437–456

14. Fischer, C.: CSP-OZ: a combination of object-Z and CSP. In: FMOODS '97: Proceedings of the IFIP TC6 WG6.1 international workshop on Formal methods for open object-based distributed systems, London, UK, UK, Chapman & Hall, Ltd. (1997) 423–438
15. Olderog, E.R., Wehrheim, H.: Specification and (property) inheritance in CSP-OZ. Sci. Comput. Program. **55**(1-3) (2005) 227–257
16. Butler, M.: Synchronisation-based Decomposition for Event-B. In: RODIN Deliverable D19 Intermediate report on methodology. (2006)
17. Butler, M.: Stepwise Refinement of Communicating Systems. Science of Computer Programming **27**(2) (September 1996) 139–173
18. Abrial, J.R.: The B-Book: Assigning programs to meanings. Cambridge University Press (1996)
19. Abrial, J.R.: Summary of Event-B Proof Obligations. http://www.docstoc.com/docs/7055755/Summary-of-Event-BProof-Obligations (March 2008)
20. Eclipse: Eclipse homepage. http://www.eclipse.org (September 2008)
21. Butler, M.: A System-based Approach to the Formal Development of Embedded Controllers for a Railway. Design Automation for Embedded Systems **6** (2002) 355–366
22. Ralph-Johan R. Back: Refinement calculus, part II: parallel and reactive programs. In: REX workshop: Proceedings on Stepwise refinement of distributed systems: models, formalisms, correctness, New York, NY, USA, Springer-Verlag New York, Inc. (1990) 67–93
23. Abadi, M., Lamport, L.: Composing Specifications. In de Bakker, J.W., de Roever, W.P., Rozenberg, G., eds.: Stepwise Refinement of Distributed Systems - Models, Formalisms, Correctness. Volume 430., Berlin, Germany, Springer-Verlag (1989) 1–41
24. Woodcock, J., Dickinson, B.: Using VDM with Rely and Guarantee-Conditions. In: Proceedings of the 2nd VDM-Europe Symposium on VDM—The Way Ahead, New York, NY, USA, Springer-Verlag New York, Inc. (1988) 434–458
25. Potet, M.L., Rouzaud, Y.: Composition and Refinement in the B-Method. In: B '98: Proceedings of the Second International B Conference on Recent Advances in the Development and Use of the B Method, London, UK, Springer-Verlag (1998) 46–65
26. Schneider, S.: The B method: an introduction. Palgrave (2001)
27. Bellegarde, F., Julliand, J., Kouchnarenko, O.: Synchronized Parallel Composition of Event Systems in B. In: ZB '02: Proceedings of the 2nd International Conference of B and Z Users on Formal Specification and Development in Z and B, London, UK, Springer-Verlag (2002) 436–457
28. Butler, M., Waldén, M.: Distributed System Development in B. Technical Report TUCS-TR-53, Turku Centre for Computer Science (14, 1996)
29. Papatsaras, A., Stoddart, B.: Global and Communicating State Machine Models in Event Driven B: A Simple Railway Case Study. In: ZB 2002:Formal Specification and Development in Z and B. Volume 2272. Springer Berlin / Heidelberg (2002) 77–100
30. Abrial, J.R., Hallerstede, S.: Refinement, Decomposition, and Instantiation of Discrete Models: Application to Event-B. Fundam. Inf. **77**(1-2) (2007) 1–28