Towards the Composition of Specifications in Event-B

Renato Silva[1,2]

School of Electronics and Computer Science
University of Southampton
Southampton, UK

Abstract

The development of a system can start with the creation of a specification. Following this viewpoint, we claim that often a specification can be constructed from the combination of specifications which can be seen as composition. Event-B is a formal method that allows modelling and refinement of systems. The combination, reuse and validation of component specifications are not currently supported in Event-B. We extend the Event-B formalism using shared event composition as an option for developing (distributed) systems. Refinement is used in the development of specifications using composed machines and we prove that properties and proof obligations of specifications can be reused to ensure valid composed specifications. The main contributions of this work are the Event-B extension to support shared event composition and refinement including the proof obligations for a composed machine.

Keywords: composition, refinement, Event-B, development of specifications, formal methods

1 Introduction

Systems can often be seen as a combination and interaction of several sub-specifications (hereafter called sub-components) where each sub-component has its own functionality aspect. This view introduces *modularity* in the system: different sub-components represent a particular functionality and changes in the sub-components are accommodated more gracefully [12] in the system specification. We use *composition* to structure specifications through

[1] R. Silva receives a Doctoral Degree Grant sponsored by Fundação Ciência e Tecnologia (FCT-Portugal). Part of this research was carried out within the European Commission ICT project 214158 DEPLOY (http://www.deploy-project.eu)

[2] Email:ras07r@ecs.soton.ac.uk

the interaction of sub-components seen as independent modules. This use of composition is not new in other formal notations: examples are [22,13,15]. Here we express how we can use (and reuse) composition for building specifications in Event-B [2] through sub-components (modules) interaction, benefiting from their properties and proof obligations (POs). The interesting part of composition involves the interaction of sub-components which usually occurs by shared state [4], shared operations [7] or a combination of both (for example, fusion composition [15]). Although sub-components have states, we mainly focus on their (visible) events similar to CSP [11,14]: we follow a *shared event composition approach* where events are synchronised in parallel.

This document is structured as follows: Sect. 2 briefly describes Event-B. Section 3 introduces the notion and properties for shared event approach. Composed machine, POs and the monotonicity property are introduced in Sect. 4. Related work, conclusions and future work are drawn in Sect. 5.

2 Event-B Language

Event-B is a formal methodology that uses mathematical techniques based on set theory and first order logic supporting system development. An abstract Event-B specification is divided into a static part called *context* and a dynamic part called *machine*. A machine *sees* as many contexts as desired. A context consists of sets s (collection of elements or a type definition), constants c and axioms $A(\dots)$ of the system. A machine contains the state (global) variables v whose values are assigned in *events*. Events that can be parameterised (local variables p) and occur when enabled by their *guards* $G(\dots)$ being true and as a result *actions* $S(\dots)$ are executed. *Invariants* $I(\dots)$ define the dynamic properties of the specification and POs are generated to verify these properties. An event *evt* is expressed by parameters p, guards $G(s,c,p,v)$ and actions $S(s,c,p,v,v')$:

$$evt \mathrel{\widehat{=}} \textbf{ANY}\ p\ \textbf{WHERE}\ G(s,c,p,v)\ \textbf{THEN}\ S(s,c,p,v,v')\ \textbf{END}.$$

When $G(s,c,p,v)$ is true then *evt* is enabled and $S(s,c,p,v,v')$ updates the set of variables v to v' (value of v after the assignment). An abstract Event-B specification can be *refined* with the introduction of more details, becoming closer to a *concrete implementation*. A context *extends* an abstract one by adding sets, constants or axioms. Machine refinement consists in refining existing events. The relation between concrete and abstract variables is given by a *gluing invariant* $J(\dots)$. POs are generated to ensure that this invariant is always preserved. New events can be added, refining *skip* and may be declared as *convergent* (the convergence is proved if each new event decreases a *variant* that must be well-founded and may be an integer or a finite set) or *anticipated* (to avoid a technical difficulty of using abstract variables in a new event during a refinement step; they must not increase the variant, only needing to decrease it when they become convergent in a further refinement [2]).

3 Shared Event Approach

Sub-component specifications that are part of a full system specification, deal with a particular part of the system being modelled. Sub-component interaction must be verified to comply with the desired behavioural semantic of the system. The interaction usually occurs as a shared state, shared event or a combination of both as described. Here we focus on the developments using shared event composition only where composition is treated as the *conjunction* of individual elements' properties: conjunction of individual invariants, conjoining variables and synchronisation of events. Consider Fig. 1(a) where

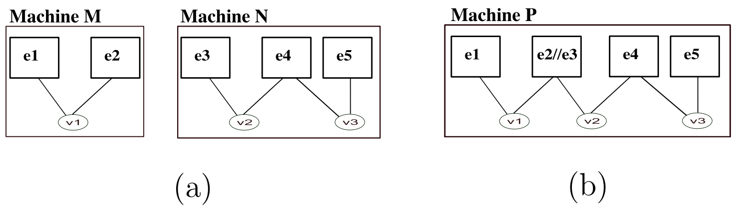

Fig. 1. Simple view of the shared event composition of M and N (a) resulting in P (b)

machine M has events *e1* and *e2* that use variable *v1*. Moreover machine N has events *e3*, *e4* and *e5* using variables *v2* and *v3*. If events *e2* and *e3* occur in parallel, they can be synchronised: machines M and N are *composed* by *sharing events*. For example, machine P in Fig. 1(b) composes *e2* from machine M and *e3* from machine *e3*: *e2* $\parallel$ *e3*. The interaction of machines M and N through their events results in a *composed event* sharing two independent variables: *v1* and *v2*. The parallel composition of events $e2$ and $e3$ from Fig. 1 is defined as Def. 3.1 [7]:

Definition 3.1 Composition of events $e2$ and $e3$ with parameter p results in:

$$\begin{aligned}
e2 \mathrel{\widehat{=}} &\textbf{ANY } p?, x \textbf{ WHERE } p? \in C \wedge G(p?, x, m) \textbf{ THEN } S(p?, x, m) \textbf{ END}\\
e3 \mathrel{\widehat{=}} &\textbf{ANY } p!, y \textbf{ WHERE } H(p!, y, n) \textbf{ THEN } T(p!, y, n) \textbf{ END}\\
e2 \parallel e3 \mathrel{\widehat{=}} &\textbf{ANY } p!, x, y \textbf{ WHERE } p! \in C \wedge G(p!, x, m) \ \wedge \ H(p!, y, n)\\
&\textbf{THEN } S(p!, x, m) \parallel T(p!, y, n) \textbf{ END}
\end{aligned}$$

where x, y, p are sets of parameters from each of the events $e2$ and $e3$. Event $e2$ has $p?$ as an input parameter and $e3$ has $p!$ as an output parameter and the resulting composition is $p!$ itself an output parameter (like in CSP). "!", "?" are used as *syntactic sugar* and are not part of the Event-B language. This property can be used to model *value-passing systems*: $e3$ sends a message to $e2$ using parameter p. Communication between parameters of type input is also possible but not between types output since this could lead to a deadlock state [7]. Event-B has the same semantic structure and refinement definitions as action systems [17]. It is possible to make a correspondence between parallel composition in CSP and an event-based view of parallel composition for action systems [9,6].

Theorem 3.2 *The shared event parallel composition of Event-B machines corresponds to CSP parallel-composition. The failure-divergence semantics of CSP can be applied to Event-B machines. The failure divergence semantics of machine M in parallel with N, $M \parallel N$ is defined as:*

$$[\![M \parallel N]\!] = [\![M]\!] \parallel [\![N]\!]$$

where $[\![M]\!]$ and $[\![N]\!]$ are the failure divergence semantics of M and N respectively. The proof of this theorem can be found in [9].

The semantics of the parallel composition of machines M and N corresponds to the set of failure-divergence for each individual machine in parallel. From the correspondence between action systems and Event-B, machines M and N can be refined independently which is one of the most important and powerful properties that shared event composition in Event-B inherits from CSP. The monotonicity property for the shared event composition in Event-B is proved by means of proof obligation in Sect. 4.4. When sub-components are composed it is desirable to define properties that relate the individual sub-components allowing interactions. These properties are expressed by adding *composition invariants* $I_{CM}(s, c, v1, \ldots, vm)$ to the composed machine constraining the variables of all machines being composed.

Definition 3.3 The invariant of the parallel composition of machines $M1$ to Mm with variables $v1$ to vm respectively is the conjunction of the individual invariants and the composition invariant $I_{CM}(s, c, v1, \ldots, vm)$:

$$I(M1 \parallel \cdots \parallel Mm) \mathrel{\widehat{=}} I_1(s, c, v1) \wedge \cdots \wedge I_m(s, c, vm) \wedge I_{CM}(s, c, v1, \ldots, vm). \qquad (1)$$

In Fig. 1, *composed machine* P has as invariant the conjunction of the individual invariants $I(M \parallel N) \mathrel{\widehat{=}} I_M(s, c, v1) \wedge I_N(s, c, v2, v3)$ plus possible composition invariant $I_{CM}(s, c, v1, v2, v3)$. In a shared event composition the sub-components have independent state space (variables are unique to each machine). Although the resulting invariant in composed machines is more complex than the original ones, due to the state space separation and conjunction of properties, the individual invariants are automatically discharged in the composed machines. We only need to deal with composition invariants. Consequently composition reasoning is simplified as there are no constraints between state spaces of sub-components.

4 Composed Machines: Composition and Refinement

We define a new construct *composed machine*, representing the shared event composition of Event-B machines. We aim to have a construct that remains reactive to changes in the sub-components. Consequently the composition is *structural*. The interaction of sub-components following a "top-down" approach, can represent a *refinement* of an existing abstraction. To formalise the composition, it is necessary to define composition and refinement POs. In the following sections, we introduce the structure of a composed machine, static checks, respective POs and prove the monotonicity property.

4.1 Structure of Composed Machines

A shared event composed machine is expressed as the parallel conjunction of sub-component properties. Machines are composed in parallel including their properties and events: $CM \mathrel{\widehat{=}} M1 \parallel \cdots \parallel Mm$ as seen in Fig. 2. Moreover:

- The composed machine variables are all the sub-component variables (v_1 from $M1$, v_2 from $M2$, ..., v_m from Mm) and are state-space disjoint.
- The invariants of the composed machine are defined as Def. 3.3.
- The composed events are defined according to Def. 3.1.

```
**COMPOSED MACHINE** $CM$ **SEES** $Ctx$
**INCLUDES** $M_1, \ldots, M_m$
**VARIABLES** $v_1, \ldots, v_m$
**INVARIANTS** $I_{CM}(s, c, v_1, v_2, \ldots, v_m)$
**EVENTS**
    $evt_{11} \mathrel{\widehat{=}} M1.evt_{11} \parallel \ldots Mm.evt_{m1}$
    ...
    $evt_{1p} \mathrel{\widehat{=}} M1.evt_{1p} \parallel \ldots Mm.evt_{mp}$
END
```

Fig. 2. Composed machine CM composing machines $M1$ to Mm seeing context Ctx

4.2 Static Checks

For the implementation of a tool for composition, composed machines need to be validated against some well-formedness conditions. We distinguish between necessary technical conditions for the composition and methodological conditions (convenient and for simplicity). The technical conditions are as follows:

- The machines to be composed belong to independent refinement chains.
- A composed event is defined by events of different sub-components.
- The same event can be synchronised and composed with different events.

The methodological conditions are:

- A composed machine is defined by at least one sub-component.
- Composed machines refinining an abstract one do not introduce new events. For simplicity we restrict adding new events. Adding events before or after the composition has a similar outcome to adding them during composition.
- A composed event is defined by at least one event.

- Variants are not required for composed machines. Only new convergent or anticipated events require variants and they are not allowed, as justified in the previous point. Consequently, we restrict anticipated event refinements.
- When the composed machine refines an abstract machine, the rules and refinement POs are applied similarly to standard machines.

An important point to address is the convergence of composed events. If we care about convergence, then the events in the included machines must be convergent. The composed events result from parallel synchronisation of include machines events. Therefore the convergence of composed events relies on the convergence of the original events. The conclusion is: if the events to be composed are convergent, then the resulting composed event is also convergent. Next we present the required POs to verify composed machines.

4.3 Proof Obligations

POs play an important role in Event-B developments. POs are generated to verify the properties of a model. For simplicity we define POs in terms of a composition of two machines M_1 and M_2 that refine machine M_0, but the rules generalise easily to the composition of n machines. Furthermore context elements in the formulas $(s, c, A(s, c))$ are not considered. The POs defined for standard machines (invariant preservation, well-definedness, refinement, etc) [2] are defined for composed machines. We simplify the composed machines POs by assuming that the POs of the individual machines hold. We define the additional POs necessary to ensure that the composed machine satisfies all the standard POs. We consider that the POs of the $M0$, M_1 and M_2 hold. The respective composition POs are described as follows.

4.3.1 Consistency

Consistency POs are required to be always verified. Consistency is expressed by the feasibility and invariant preservation POs for each event. The feasibility proof obligation for the composed event $evt1 \parallel evt2$ is $FIS_{evt1 \parallel evt2}$.

Theorem 4.1 *The individual FIS PO for each event can be reused for proving feasibility for each composed event and that is enough to verify this property. From [2]:*

$$FIS_{evt1}: \quad I_1(v_1) \wedge G_1(p_1, v_1) \vdash \exists v_1' \cdot (S_1(p_1, v_1, v_1')) \tag{2}$$

$$FIS_{evt2}: \quad I_2(v_2) \wedge G_2(p_2, v_2) \vdash \exists v_2' \cdot (S_2(p_2, v_2, v_2')) \tag{3}$$

$$\begin{aligned} FIS_{evt1 \parallel evt2}: \quad & I_{CM}(v_0, v_1, v_2) \wedge I_1(v_1) \wedge I_2(v_2) \wedge G_1(p_1, v_1) \wedge G_2(p_2, v_2) \\ & \vdash \exists v_1', v_2' \cdot (S_1(p_1, v_1, v_1') \wedge S_2(p_2, v_2, v_2')). \end{aligned} \tag{4}$$

Assume: FIS_{evt1} *and* FIS_{evt2}.
Prove: $FIS_{evt1 \parallel evt2}$.

Proof. Assume the hypotheses of $FIS_{evt1\|evt2}$. Prove: $\exists v_1', v_2' \cdot (S_1(p_1, v_1, v_1') \wedge S_2(p_2, v_2, v_2'))$. The proof proceeds as follows:

$$\begin{array}{ll} \exists v_1' \cdot (S_1(p_1, v_1, v_1')) \wedge \exists v_2' \cdot (S_2(p_2, v_2, v_2')) & \{\text{disjoint v1 and v2}\} \\ \Leftarrow (FIS_{evt1} \wedge FIS_{evt2}). & \{(2),(3)+ \text{ hypotheses of } (4)\} \end{array}$$

□

In the composed machine, invariant preservation PO INV_{CM} corresponds to the invariant preservation in all events. The invariant preservation PO for the composed event $evt1 \parallel evt2$ is $INV_{evt1\|evt2}$.

Theorem 4.2 *For each invariant i from the set of invariants I in a composed machine, composition invariant $I_{CM}(v_0, v_1, v_2)$ needs to be verified. From [2]:*

$$\begin{aligned} INV_{evt1}: \quad & I_1(v_1) \wedge G_1(p_1, v_1) \wedge S_1(p_1, v_1, v_1') \vdash i_1(v_1') && (5) \\ INV_{evt2}: \quad & I_2(v_2) \wedge G_2(p_2, v_2) \wedge S_2(p_2, v_2, v_2') \vdash i_2(v_2') && (6) \\ INV_{evt1\|evt2}: \quad & I_{CM}(v_0, v_1, v_2) \wedge I_1(v_1) \wedge I_2(v_2) \wedge G_1(p_1, v_1) \wedge G_2(p_2, v_2) \\ & \wedge S_1(p_1, v_1, v_1') \wedge S_2(p_2, v_2, v_2') \vdash i_1(v_1') \wedge i_2(v_2') \wedge i_{CM}(v_0, v_1', v_2') && (7) \end{aligned}$$

Assume: INV_{evt1} and INV_{evt2}.
Prove: $INV_{evt1\|evt2}$.

Proof. Assume the hypotheses of $INV_{evt1\|evt2}$. Prove: $i_1(v_1') \wedge i_2(v_2') \wedge i_{CM}(v_0, v_1', v_2')$. The proof proceeds as follows:

$$\begin{array}{ll} i_1(v_1') \wedge i_2(v_2') \wedge i_{CM}(v_0, v_1', v_2') & \\ \Leftarrow INV_{evt1} \wedge INV_{evt2} \wedge i_{CM}(v_0, v_1', v_2'). & \{(5),(6) \text{ and hypotheses of } (7)\} \end{array}$$

□

Well-definedness for expressions (guards, actions, invariants, etc) needs to be verified. These are verified by means of POs in Event-B [3]. For composed machines, well-definedness POs are only generated for $I_{CM}(v_0, v_1, v_2)$. Other expressions are verified in the individual machines.

4.3.2 Refinement

Refinement POs are required when the composed machine refines an abstract machine. Machine M_0 with variables v_0, invariant $I_0(v_0)$ and abstract event evt_0 is refined by composed machine CM defined by machines M_1 with variables w_1, invariant $I_1(w_1)$, event evt_1 and M_2 (w_2 ; $I_2(w_2)$; evt_2) and composition invariant $J_{CM}(v_0, w_1, w_2)$. The composed event $evt1 \parallel evt2$ refines the abstract event evt_0. A general refinement PO (REF$_{evti}$) for a machine M refining event evt follows from:

$$REF_{evti} \mathrel{\widehat{=}} I_i(v_i) \wedge J_i(v_i, w_i) \wedge H_i(q_i, w_i) \wedge T_i(q_i, w_i, w_i') \vdash \exists v_i' \cdot G_i(v_i) \wedge S_i(p_i, v_i, v_i') \wedge J_i(v_i', w_i'). \qquad (8)$$

Theorem 4.3 *For each composed event evt1 $\|$ evt2, refining abstract event evt0 through (gluing) composition invariant in a composed machine, the refinement REF PO consists in proving the guard strengthening of abstract guards, proving the simulation of the abstract variables (v_0') and preserving the gluing invariant ($J_{CM}(v_0', w_1', w_2')$). From [2] and (8):*

$$
\begin{aligned}
INV_{evt1}: &\quad I_1(w_1) \wedge H_1(q_1, w_1) \wedge T_1(q_1, w_1, w_1') \vdash i_1(w_1') && (9)\\
INV_{evt2}: &\quad I_2(w_2) \wedge H_2(q_2, w_2) \wedge T_2(q_2, w_2, w_2') \vdash i_2(w_2') && (10)\\
REF_{evt0 \sqsubseteq (evt1 \| evt2)}: &\quad I_0(v_0) \wedge I_1(w_1) \wedge I_2(w_2) \wedge J_{CM}(v_0, w_1, w_2)\\
&\quad \wedge H_1(q_1, w_1) \wedge H_2(q_2, w_2) \wedge T_1(q_1, w_1, w_1') \wedge T_2(q_2, w_2, w_2')\\
&\quad \vdash \exists v_0' \cdot G_0(p_0, v_0) \wedge S_0(p_0, v_0, v_0') \wedge I_1(w_1') \wedge I_2(w_2') \wedge J_{CM}(v_0', w_1', w_2').
\end{aligned}
$$

Assume: INV_{evt1} (9) *and* INV_{evt2} (10).
Prove: $REF_{evt0 \sqsubseteq (evt1 \| evt2)}$.

Proof. Assume the hypotheses of $REF_{evt0 \sqsubseteq (evt1 \| evt2)}$. Prove: $\exists v_0' \cdot G_0(p_0, v_0) \wedge S_0(p_0, v_0, v_0') \wedge I_1(w_1') \wedge I_2(w_2') \wedge J_{CM}(v_0', w_1', w_2')$. The proof proceeds as follows:

$$
\begin{aligned}
& G_0(p_0, v_0) \wedge I_1(w_1') \wedge I_2(w_2') \\
& \wedge \exists v_0' \cdot (S_0(p_0, v_0, v_0') \wedge J_{CM}(v_0', w_1', w_2')) && \{\wedge \text{ goal; } v_0, w_1', w_2' \text{ are free variables}\}\\
\equiv\ & G_0(p_0, v_0) \wedge \exists v_0' \cdot (S_0(p_0, v_0, v_0') \wedge J_{CM}(v_0', w_1', w_2')) && \{\text{from (9) + (10) for each } i_1(w_1'), i_2(w_2')\}
\end{aligned}
$$

□

These are the required POs to verify composed machines. Next we show that composed machines are monotonic which allows to further refine individual machines preserving composition.

4.4 Monotonicity of Shared Event Composition for Composed Machines

An important property of the shared event composition in Event-B is *monotonicity*. We prove it by means of refinement POs confirming the result described by Butler [9] using actions systems and CSP. Figure 3 shows abstract component specification $M1$ composed with other component specification $N1$, creating a composed model $M1 \| N1$. $M1$ is refined by $M2$ and $N1$ by $N2$ respectively. Once we compose component specifications $M1$ and $N1$, discharge the required composed POs, $M1$ and $N1$ can be refined individually while the composition properties are preserved without the need to recompose refinements $M2$ and $N2$. We want to formally prove the monotonicity

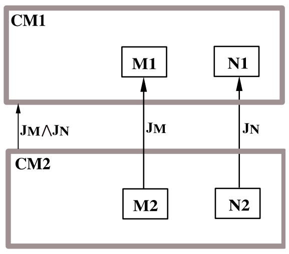

Fig. 3. Refinement of composed machine $CM1 \mathrel{\hat{=}} M1 \| N1$ by $CM2 \mathrel{\hat{=}} M2 \| N2$

property through refinement POs between composed machines. Therefore if the refinement POs hold between $CM1$ and $CM2$, we can say that $CM2$ refines $CM1$: $CM1 \sqsubseteq CM2$. The gluing invariant of the refinement between M1 and M2 is expressed as $J_M(v_M, w_M)$ relating the states of M1 and M2: $M1 \sqsubseteq_{J_M} M2$. We can derive the refinement PO between $M2$ and $M1$ for the concrete event evt_{M2} refining abstract event evt_{M1}.

$$\begin{aligned} REF_{evt_{M1} \sqsubseteq evt_{M2}}: \quad & I_M(v_M) \wedge J_M(v_M, w_M) \wedge G_M(p_M, v_M) \wedge H_M(q_M, w_M) \\ & \wedge S_M(p_M, v_M, v'_M) \wedge T_M(q_M, w_M, w'_M) \\ & \vdash \exists v'_M \cdot G_M(p_M, v_M) \wedge S_M(p_M, v_M, v'_M) \wedge J_M(v'_M, w'_M). \end{aligned} \tag{11}$$

The refinement PO between $N2$ and $N1$ is similar. We refine an abstract event in CM1 by a concrete one in CM2 and verify that the refinement POs for each individual machine hold for the composition. Event evt_{M1} from machine $M1$ and event evt_{N1} from machine N1 are composed, resulting in the abstract composed event $evt_{M1} \parallel evt_{N1}$ in $CM1$ from Fig. 3. The gluing invariant relating the states of $CM1$ and $CM2$ is expressed as the conjunction of the gluing invariants between ($M1$ and $M2$) and ($N1$ and $N2$):

$$J_{CM}(v_M, v_N, w_M, w_N) = J_M(v_M, w_M) \wedge J_N(v_N, w_N) \tag{12}$$

Theorem 4.4 *The refinement POs for composed machines is expressed as the conjunction of the refinement POs for the individual machines. Therefore the monotonicity property holds if the refinement POs of individual machines hold. The refinement PO between concrete composed event $evt_{M2} \parallel evt_{N2}$ and abstract composed event $evt_{M1} \parallel evt_{N1}$ is expressed as:*

$$\begin{aligned} REF_{(evt_{M1} \parallel evt_{N1}) \sqsubseteq (evt_{M2} \parallel evt_{N2})}: \quad & I_M(v_M) \wedge I_N(v_N) \wedge J_{CM}(v_M, v_N, w_M, w_N) \wedge H_M(q_M, w_M) \\ & \wedge H_N(q_N, w_N) \wedge T_M(q_M, w_M, w'_M) \wedge T_N(q_N, w_N, w'_N) \\ & \vdash \exists v'_M, v'_N \cdot G_M(p_M, v_M) \wedge G_N(p_N, v_N) \wedge S_M(p_M, v_M, v'_M) \\ & \wedge S_N(p_N, v_N, v'_N) \wedge J_{CM}(v'_M, v'_N, w'_M, w'_N). \end{aligned} \tag{13}$$

Assume: $REF_{evt_{M1} \sqsubseteq evt_{M2}}$ and $REF_{evt_{N1} \sqsubseteq evt_{N2}}$.
Prove: $REF_{(evt_{M1} \parallel evt_{N1}) \sqsubseteq (evt_{M2} \parallel evt_{N2})}$.

Proof. Assume the hypotheses of $REF_{(evt_{M1} \parallel evt_{N1}) \sqsubseteq (evt_{M2} \parallel evt_{N2})}$. Prove: $\exists v'_M, v'_N \cdot G_M(p_M, v_M) \wedge G_N(p_N, v_N) \wedge S_M(p_M, v_M, v'_M) \wedge S_N(p_N, v_N, v'_N) \wedge J_{CM}(v'_M, v'_N, w'_M, w'_N)$. The proof proceeds as follows:

$$\begin{aligned} & \exists v'_M, v'_N \cdot G_M(p_M, v_M) \wedge G_N(p_N, v_N) \\ & \quad \wedge S_M(p_M, v_M, v'_M) \wedge S_N(p_N, v_N, v'_N) \\ & \quad \wedge J_M(v'_M, w'_M) \wedge J_N(v'_N, w'_N) && \{\text{expanding } J_{CM} \text{ from (12)}\} \\ & \equiv \exists v'_M \cdot G_M(v_M) \wedge S_M(p_M, v_M, v'_M) \wedge J_M(v'_M, w'_M) \\ & \quad \wedge \exists v'_N \cdot G_N(v_N) \wedge S_N(p_N, v_N, v'_N) \wedge J_N(v'_N, w'_N) && \{\text{disjoint } v'_M, v'_N\} \\ & \Leftarrow REF_{evt_{M1} \sqsubseteq evt_{M2}} \wedge REF_{evt_{N1} \sqsubseteq evt_{N2}} && \{(11) + \text{hypotheses of } (13)\} \end{aligned}$$

□

We also need to prove the monotonicity for single (non-composed) events that appear at both levels of abstraction. We shall prove it using machines $M1$ and $CM2$. In this case, the gluing invariant described in (12) does not use neither the variables (v_N) neither the invariants(I_N) neither events (evt_{N1}) from $N1$. Therefore it can be simplified and rewritten as:

$$J_{CM}(v_M, w_M, w_N) = J_M(v_M, w_M) \wedge J_N(w_N) \tag{14}$$

Theorem 4.5 *An individual event evt_{M1} in machine $M1$ is refined by a composed event $evt_{M2} \parallel evt_{N2}$ in composed machine $CM2$. The monotonicity is preserved if the refinement PO between $M1$ and $M2$ hold in conjunction with the gluing invariant preservation PO for the composed event $evt_{M2} \parallel evt_{N2}$. The refinement PO between concrete composed event $evt_{M2} \parallel evt_{N2}$ and abstract non-composed event evt_{M1}:*

$$\begin{aligned}REF_{evt_{M1} \sqsubseteq (evt_{M2} \parallel evt_{N2})} : \quad & I_M(v_M) \wedge J_{CM}(v_M, w_M, w_N) \wedge H_M(q_M, w_M) \wedge H_N(q_N, w_N) \\ & \wedge T_M(q_M, w_M, w'_M) \wedge T_N(q_N, w_N, w'_N) \\ & \vdash \exists v'_M \cdot G_M(p_M, v_M) \wedge S_M(p_M, v_M, v'_M) \wedge J_{CM}(v'_M, w'_M, w'_N). \end{aligned} \tag{15}$$

Assume: $REF_{evt_{M1} \sqsubseteq evt_{M2}}$ *and* $INV_{evt_{M2} \parallel evt_{N2}}$ *(based on (7)).*
Prove: $REF_{evt_{M1} \sqsubseteq (evt_{M2} \parallel evt_{N2})}$.

Proof. Assume the hypotheses of $REF_{evt_{M1} \sqsubseteq (evt_{M2} \parallel evt_{N2})}$ and the hypotheses of $INV_{evt_{M2} \parallel evt_{N2}}$.

Prove: $\exists v'_M \cdot G_M(p_M, v_M) \wedge S_M(p_M, v_M, v'_M) \wedge J_{CM}(v'_M, w'_M, w'_N)$. The proof proceeds as follows:

$$\begin{aligned} & \exists v'_M \cdot G_M(p_M, v_M) \wedge S_M(p_M, v_M, v'_M) \wedge J_M(v'_M, w'_M) \wedge J_N(w'_N) && \{\text{expanding } J_{CM} \text{ by } (14)\} \\ & \equiv \exists v'_M \cdot (G_M(p_M, v_M \wedge S_M(p_M, v_M, v'_M) \wedge J_M(v'_M, w'_M)) \wedge J_N(w'_N) && \{\text{free } v'_N\} \\ & \Leftarrow REF_{evt_{M1} \sqsubseteq evt_{M2}} \wedge J_N(w'_N) && \{(11)\text{+hypotheses of } (15)\} \\ & \Leftarrow REF_{evt_{M1} \sqsubseteq evt_{M2}} \wedge INV_{evt_{M2} \parallel evt_{N2}} && \{(7)\} \end{aligned}$$

□

New events can be added during refinement, respecting the refinement POs. The refinement PO proof for new events is similar to the previous cases but applied to a single event refined by a composed event. Due to the lack of space we do not present it here.

5 Related Work, Conclusions and Future Work

Composition allows the interaction of sub-components. Back [16], Abadi and Lamport[1] studied the interaction of components through shared variable composition. Jones [21] 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 Z, composition can be achieved by combining schemas [20] where variables within the same scope cannot have identical names or by views [12] allowing the development of partial specifications that can interact through invariants that relate their state or by operations' synchronisation. Although systems are developed in single machines in classical B, Bellergarde et at [5] suggest a composition by rearranging separated machines and synchronising their operations under feasibility conditions. The behaviour of a component composition is seen as a labelled transition system using weakest preconditions, where a set of authorised transitions are defined. The objective is to verify the refinement of synchronised parallel composition between components but it is limited to finite state transitions and a finite number of components. This work differs from ours as it uses a labelled transition system allowing variable sharing while we use synchronisation and communication in the CSP style. Butler and Walden [8] discuss 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 sub-systems in a parallel composition may be refined independently. This work is closely related to our work with similar underlying semantics and notion of refinement based on CSP. Abrial et al [4] propose a state-based decomposition for Event-B introducing the notion of shared variables and external events. Although it allows variable sharing, this approach is also monotonic but its respective nature is more suitable for parallel programs [10].

Our Event-B composition is based on the close relation between action systems and Event-B plus the correspondence between action systems and CSP [9]. A methodology for composition is defined including the verification of properties through the generation of POs. We extend Event-B to support shared event composition, allowing combination and reuse of existing sub-components through *composed machines* and we prove it to be monotonic. Refinement in a “top-down” style for developing specifications is allowed. Sub-components interact through parameters by value-passing and can be further refined. POs of included machines are reused to discharge composition POs. *Composition invariants* can be added relating the state space of included machines, generating additional POs. From our experience, these POs “suggest” the (possible) modifications to be applied to the included machines, ensuring the invariant preservation by the composed events. Refinement of anticipated events is currently not allowed. We shall study this option in the future, lifting the restriction of variants usage. This approach seems suitable for modelling (distributed) systems resulting from the exploration of specifications’ composition. We do not address the step corresponding to the translation of this composition to an implementation. This study needs to be carried out in the future. A tool has been developed to support composition in the Rodin platform [18]. Some case studies have been applying composition with success, in particular for distributed systems as part of *decomposition* [19].

References

[1] Martín Abadi and Leslie Lamport. Composing Specifications. In J. W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Stepwise Refinement of Distributed Systems - Models, Formalisms, Correctness*, volume 430, pages 1–41, Berlin, Germany, 1989. Springer-Verlag.

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

[3] Jean-Raymond Abrial, Michael Butler, Stefan Hallerstede, Thai Son Hoang, Farhad Mehta, and Laurent Voisin. Rodin: An Open Toolset for Modelling and Reasoning in Event-B. *International Journal on Software Tools for Technology Transfer (STTT)*, April 2010.

[4] Jean-Raymond Abrial and Stefan Hallerstede. Refinement, Decomposition, and Instantiation of Discrete Models: Application to Event-B. *Fundam. Inf.*, 77(1-2):1–28, 2007.

[5] Françoise Bellegarde, Jacques Julliand, and Olga Kouchnarenko. 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*, pages 436–457, London, UK, 2002. Springer-Verlag.

[6] Michael Butler. Stepwise Refinement of Communicating Systems. *Science of Computer Programming*, 27(2):139–173, September 1996.

[7] Michael Butler. 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*, pages 221–241, 1997.

[8] Michael Butler and Marina Waldén. Distributed System Development in B. Technical Report TUCS-TR-53, Turku Centre for Computer Science, 14, 1996.

[9] Michael J. Butler. *A CSP Approach to Action Systems.* PhD thesis, Oxford University, 1992.

[10] Thai Hoang and Jean-Raymond Abrial. Event-B Decomposition for Parallel Programs. *Abstract State Machines, Alloy, B and Z*, pages 319–333, 2010.

[11] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice Hall International Series in Computer Science, 1985.

[12] Daniel Jackson. Structuring Z specifications with views. *ACM Trans. Softw. Eng. Methodol.*, 4(4):365–389, 1995.

[13] Cliff B. Jones. Wanted: a compositional approach to concurrency. In *Programming methodology*, pages 5–15. Springer-Verlag New York, Inc., New York, NY, USA, 2003.

[14] Carroll Morgan. Of wp and CSP. In *Beauty is our business: a birthday salute to Edsger W. Dijkstra*, pages 319–326. Springer-Verlag New York, Inc., New York, NY, USA, 1990.

[15] Michael Poppleton. The Composition of Event-B Models. In *ABZ2008: Int. Conference on ASM, B and Z*, volume 5238, pages 209–222. Springer LNCS, September 2008.

[16] 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*, pages 67–93, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

[17] Ralph-Johan R. Back and R. Kurki-Suonio. Decentralization of Process Nets with Centralized Control. In *PODC '83: Proceedings of the second annual ACM symposium on Principles of distributed computing*, pages 131–142, New York, NY, USA, 1983. ACM.

[18] Renato Silva and Michael Butler. Parallel Composition Using Event-B. http://wiki.event-b.org/index.php/Parallel_Composition_using_Event-B, July 2009. Online; accessed 27-July-2010.

[19] Renato Silva, Carine Pascal, Thai Son Hoang, and Michael Butler. Decomposition Tool for Event-B. *Software: Practice and Experience*, 41(2):199–208, February 2011.

[20] J. M. Spivey. *The Z Notation: a Reference Manual.* Prentice-Hall, Inc., 1989.

[21] Jim Woodcock and B. Dickinson. Using VDM with Rely and Guarantee-Conditions. In *Proceedings of the 2nd VDM-Europe Symposium on VDM—The Way Ahead*, pages 434–458, New York, NY, USA, 1988. Springer-Verlag New York, Inc.

[22] Pamela Zave and Michael Jackson. Conjunction as Composition. *ACM Trans. Softw. Eng. Methodol.*, 2(4):379–411, 1993.