Abstract. The construction of specifications is often a combination of smaller sub-components. Composition and decomposition are techniques that support reuse and allow us to formally combine sub-components through refinement steps while reusing their properties. Sub-components can result from a design or architectural goal and a refinement framework should allow further parallel development over the sub-components. We propose the definition of composition and decomposition in the Event-B formalism following a shared event approach where sub-components interact via synchronisation over shared events and shared states are not allow. We define the necessary proof obligations to ensure a valid composition or decomposition. We also show that shared event composition preserves refinement proofs for sub-components, that is, in order to maintain refinement of compositions, it is sufficient to prove refinement between corresponding sub-components. A case study applying these two techniques is illustrated using Rodin, the Event-B toolset.

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

1 Introduction

The development of specifications in a “top-down” style starts with an abstract model of the envisaged system. 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 [1] in the system specification. We use composition to structure specifications through the interaction of sub-components seen as independent modules. This use of composition is not new in other formal notations: examples are [2,3,4]. Here we express how we can use (and reuse) composition for building specifications in Event-B [5] 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 [6], shared operations [7] or a combination of both (for example, fusion.
composition [4]). Although sub-components have states, we mainly focus on their (visible) events similar to CSP [8,9]; we follow a shared event composition approach where events are synchronised in parallel. Decomposition is motivated by the possibility of breaking a complex problem or system into parts that are easier to conceive, manage and maintain. The partition of a model into sub-components can also be seen as a design/architectural decision and the further development of the sub-components in parallel is possible. Besides alleviating the complexity for large systems and respective proofs, decomposition allows team development in parallel over the same model which is very attractive in the industrial environment. Moreover the proof obligations of the original (non-decomposed) model can be reused by the sub-components. We present in more detail the shared event approach applied to composition and decomposition. Moreover, the proof obligations to ensure a valid composition are expressed including the possibility to reuse the sub-components properties. The monotonicity property for composition is proved by means of refinement proof obligations. We see decomposition as the inverse operation of composition and therefore we can reuse its properties to decompose systems. A guideline in how to apply a shared event decomposition is presented and a case study is illustrated to highlight the use of this technique. The models are developed in the Rodin [10], which is a toolset for Event-B [5,11].

This document is structured as follows: Section 2 gives an overview of the Event-B formal method. Section 3 introduces the notion and motivation for shared event approach for composition and decomposition. Composed machines, properties, proof obligations are described in Sect. 4. A guideline in how to use decomposition is presented in Sect. 5. Section 6 illustrates the application of composition and decomposition to a distributed system case study: file access system. Related work, conclusions and future work are drawn in Section 7.

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 with abstract specification. 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(…) 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) occur when enabled by their guards G(…) being true and as a result actions S(…) are executed. Invariants I(…) define the dynamic properties of the specification and POs are generated to verify that these properties are always maintained. An event \(\text{evt} \) is expressed by parameters \(p\), by guards \(G(s, c, p, v)\) and actions \(S(s, c, p, v, v')\):

\[
\text{evt} \equiv \text{ANY } p \text{ WHERE } G(s, c, p, v) \text{ THEN } S(s, c, p, v, v') \text{ END.}
\]

When the guard \(G(s, c, p, v)\) is true then the event \(\text{evt} \) is enabled and therefore the action \(S(s, c, p, v, v')\) updates the set of variables \(v\) to \(v'\) (value of \(v\) after
An abstract Event-B specification can be refined with the introduction of more details and becoming closer to a concrete implementation. A context extends an abstract context by adding sets, constants or axioms. The abstract context properties are still assumed. Refinement of a machine consists of refining existing events. The relation between variables in the concrete and abstract model is given by a gluing invariant \( J(\ldots) \). POs are generated to ensure that this invariant is preserved in the concrete model. New events can be added, refining \textit{skip} which may be declared as convergent, meaning they do not cause divergence. The convergence is proved if each new event decreases a \textit{variant}. The variant must be well-founded and may be an integer or a finite set.

### 3 Shared Event Approach

The shared event approach seems suitable for the development of distributed systems\[7]\: sub-components interact through synchronised events in parallel; moreover sub-components can communicate using shared parameters which is useful for modelling message broadcasting systems.

#### 3.1 Shared Event Composition

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. Here we focus on the developments using shared event composition where individual elements’ properties are conjoined: \textit{conjunction} of individual invariants, \textit{conjoining} variables and \textit{synchronisation} of events.

![Shared event composition of \( M_1 \) and \( M_2 \) (a) resulting in \( M \) (b)](image-url)
Consider Fig. 1 where machine $M_1$ has events $e_1$ and $e_2$ that use variable $v_1$. Moreover, machine $M_2$ has events $e_3$, $e_4$, and $e_5$ using variables $v_2$ and $v_3$. If events $e_2$ and $e_3$ occur in parallel, they can be synchronised; machines $M_1$ and $M_2$ are composed by sharing events. In Fig. 1, machine $M$ is the result of the composition of machines $M_1$ and $M_2$ where $e_2$ from machine $M_1$ and $e_3$ from machine $M_3$ are composed: $e_2 \parallel e_3$. The interaction of machines $M_1$ and $M_2$ through their events results in a composed event sharing two independent variables: $v_1$ and $v_2$.

A general definition for the parallel composition of events $e_2$ and $e_3$ is defined as Def. 1 [7]:

Definition 1. Composition of events $e_2$ and $e_3$ with parameter $p$ results in:

$$
e_2 \triangleq \text{ANY } p?, x \text{ WHERE } p? \in C \land G(p?, x, m) \text{ THEN } S(p?, x, m) \text{ END}
$$

$$
e_3 \triangleq \text{ANY } p!, y \text{ WHERE } H(p!, y, n) \text{ THEN } T(p!, y, n) \text{ END}
$$

$$
e_2 \parallel e_3 \triangleq \text{ANY } p!, x, y \text{ WHERE } p! \in C \land G(p!, x, m) \land H(p!, y, n) \text{ THEN } S(p!, x, m) \parallel T(p!, y, n) \text{ END}
$$

where $x, y, p$ are sets of parameters from each of the events $evt_1$ and $evt_2$. Event $evt_1$ has $p?$ as an input parameter and $evt_2$ has $p!$ as an output parameter and the resulting composition is $p!$ itself an output parameter (like in CSP). This property can be used to model message-passing systems: $e_3$ sends a message to $e_2$ using the parameter $p$. Communication between input type parameters is also possible but not with both output parameters since this could lead to a deadlock state [7].

Action systems [12] provides a general description of reactive systems, capable of modelling terminating, aborting and infinitely repeating systems. Event-B is inspired in action systems and can be seen as a realisation of actions systems but using a combination of logic and mathematics as a formal language. Both formalisms share the same refinement semantics. Therefore we claim that Event-B has the same semantic 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 [13,14].

Theorem 1. The shared event parallel composition of actions systems corresponds to the CSP parallel-composition. The failure-divergence semantics of CSP can be applied to action systems. The failure divergence semantics of action system $M$ 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 [13].

The semantics of the parallel composition of action systems $M$ and $N$ corresponds to the set of failure-divergence for each individual action system in parallel. From the correspondence between action systems and Event-B, $M$ 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. 4.3.

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 \( I_{CM}(s, c, v_1, \ldots, v_m) \) to the composed machine constraining the variables of all machines being composed.

**Definition 2.** The invariant of the parallel composition of machines \( M_1 \) to \( M_n \) with variables \( v_1 \) to \( v_n \) respectively is the conjunction of the individual invariants and the composition invariant \( I_{CM}(s, c, v_1, \ldots, v_n) \):

\[
\begin{align*}
I(M_1 \parallel \cdots \parallel M_m) & = I_1(s, c, v_1) \land \cdots \land I_m(s, c, v_m) \land I_{CM}(s, c, v_1, \ldots, v_n). 
\end{align*}
\] (1)

In Fig. 1 composed machine \( M \) has as invariant the conjunction of the individual invariants \( I(A \parallel B) = I_A(s, c, v_1) \land I_B(s, c, v_2, v_3) \) plus possible composition invariant \( I_{CM}(s, c, v_1, v_2, v_3) \). In a shared event composition the sub-components have independent state space (variables are unique to each machine). Consequently composition reasoning is simplified as there are no constraints between state spaces of sub-components.

### 3.2 Shared Event Decomposition

Decomposition can be seen as the inverse process of composition: after some refinements a larger model may be decomposed into smaller components. This step might be a consequence of complexity or just as an architectural decision. The shared event approach is also used: events are shared between sub-components and variable sharing is not allowed. Butler [15] proposes a shared event decomposition for Event-B inspired by CSP and action systems with event sharing as seen in Fig. 2. We follow that work in our approach. Events using variables allocated to different sub-components (\( e_2 \) shares \( v_1 \) and \( v_2 \)) must be split. The part corresponding to each variable (\( e_2' \) and \( e_2'' \)) is used to create partial versions of the original event. After the decomposition, the individual machines can be further refined since the composition relation holds. The possible recomposition of the sub-components (or their refinements) is a refinement of the original composed component although this step should never be required in practice. Figure 3 shows the decomposition of \( M_1 \) into \( M_3 \) and \( M_4 \) that are further refined into \( M_3 \) and \( M_4 \) respectively. At this stage a possible recomposition of \( M_3 \) and \( M_4 \) into \( cM_2 \) should be proved to be a refinement of \( M_1 \).

### 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 follows a “top-down” approach, representing 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, 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: $CM \equiv M_1 \parallel \cdots \parallel M_m$ as seen in Fig. 2. Moreover:

- The composed machine variables are all the sub-component variables ($v_1$ from $M_1$, $v_2$ from $M_2$, \ldots, $v_m$ from $M_m$) and are state-space disjoint.
- The invariants of the composed machine are defined as Def. 2.
- The composed events are defined according Def. 1.

When a composed machine is used as a combination of composition and refinement, it refines an abstract model and just like in an ordinary machine, abstract events must be refined. For instance, a composed machine $CM$ resulting from the parallel composition of $M_1 \ldots M_m$ and refining abstract machine $M_0$ can be expressed as $M_0 \subseteq CM \equiv M_0 \subseteq M_1 \parallel \cdots \parallel M_m$. Next we present the required POs to verify composed machines.

4.2 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 are [5].
Fig. 3. Decomposition, Recomposition and Refinement

**COMPOSED MACHINE CM**
- 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
  - $evt_{11} \equiv M_1.evt_{11} \parallel \ldots M_m.evt_{11}$
  - $\ldots$
  - $evt_{1p} \equiv M_1.evt_{1p} \parallel \ldots M_m.evt_{1p}$
- END

Fig. 4. Composed machine $CM$ composing machines $M1$ to $Mm$ seeing context $Ctx$

- Consistency: Invariant Preservation (INV) and Feasibility (FIS)
- Refinement: Guard Strengthening (GRD), Simulation/Refinement (SIM) and Gluing Invariant Preservation (INV)
- Variant: Numeric Variant (NAT), Numeric Variant Decreasing (VAR), Finite Set Variant (FIN)
- Well-Definedness (WD)

These POs also are defined for composed machines except the ones related with variant (no variant 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 $M0$, $M1$ and $M2$ hold. The respective composition POs are described as follows.

**Consistency** Consistency POs are required to be always verified. The feasibility proof obligation for the composed event $evt1 \parallel evt2$ is $FIS_{evt1|evt2}$. 
Theorem 2. 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 [5]:

\[ FIS_{evt1} : I_1(v_1) \land G_1(p_1, v_1) \vdash \exists v'_1 \cdot (S_1(p_1, v_1, v'_1)) \]  
(2)  

\[ FIS_{evt2} : I_2(v_2) \land G_2(p_2, v_2) \vdash \exists v'_2 \cdot (S_2(p_2, v_2, v'_2)) \]  
(3)  

\[ FIS_{evt1||evt2} : I_{CM}(v_0, v_1, v_2) \land I_1(v_1) \land I_2(v_2) \land G_1(p_1, v_1) \land G_2(p_2, v_2) \]  
\[ \vdash \exists v'_1, v'_2 \cdot (S_1(p_1, v_1, v'_1) \land S_2(p_2, v_2, v'_2)). \]  
(4)  

Assume: \( FIS_{evt1} \) and \( FIS_{evt2} \).

Prove: \( FIS_{evt1||evt2} \).

Proof. Assume the hypotheses of \( FIS_{evt1||evt2} \).

\[ I_{CM}(v_0, v_1, v_2) \]  
(5)  

\[ I_1(v_1) \land G_1(p_1, v_1) \]  
(6)  

Prove: \( \exists v'_1, v'_2 \cdot (S_1(p_1, v_1, v'_1) \land S_2(p_2, v_2, v'_2)) \). The proof proceeds as follows:

\[ \exists v'_1, v'_2 \cdot (S_1(p_1, v_1, v'_1) \land S_2(p_2, v_2, v'_2)) \]
\[ \equiv \exists v'_1 \cdot (S_1(p_1, v_1, v'_1)) \land \exists v'_2 \cdot (S_2(p_2, v_2, v'_2)) \quad \text{(disjoint v1 and v2)} \]
\[ \equiv (FIS_{evt1} \land FIS_{evt2}). \]  
\[ \{2 + 3 + 4\} \]

Another consistency PO is invariant preservation. In the composed machine, invariant preservation PO \( INV_{CM} \) corresponds to the invariant preservation in all events from the individual machines that are composed. The invariant preservation proof obligation for the composed event \( \text{evt1} \parallel \text{evt2} \) is \( INV_{evt1||evt2} \).

Theorem 3. 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 [5]:

\[ INV_{evt1} : I_1(v_1) \land G_1(p_1, v_1) \land S_1(p_1, v_1, v'_1) \vdash i_1(v'_1) \]  
(7)  

\[ INV_{evt2} : I_2(v_2) \land G_2(p_2, v_2) \land S_2(p_2, v_2, v'_2) \vdash i_2(v'_2) \]  
(8)  

\[ INV_{evt1||evt2} : I_{CM}(v_0, v_1, v_2) \land I_1(v_1) \land I_2(v_2) \land G_1(p_1, v_1) \land G_2(p_2, v_2) \land S_1(p_1, v_1, v'_1) \land S_2(p_2, v_2, v'_2) \]  
\[ \vdash i_1(v'_1) \land i_2(v'_2) \land i_{CM}(v_0, v'_1, v'_2) \]

Assume: \( INV_{evt1} \) and \( INV_{evt2} \).

Prove: \( INV_{evt1||evt2} \).

Proof. Assume the hypotheses of \( INV_{evt1||evt2} \).

\[ I_{CM}(v_0, v_1, v_2) \]  
(9)  

\[ I_1(v_1) \land G_1(p_1, v_1) \land S_1(p_1, v_1, v'_1) \]  
(10)  

Prove: \( i_1(v'_1) \land i_2(v'_2) \land i_{CM}(v_0, v'_1, v'_2) \). The proof proceeds as follows:

\[ i_1(v'_1) \land i_2(v'_2) \land i_{CM}(v_0, v'_1, v'_2) \]
\[ \equiv INV_{evt1} \land INV_{evt2} \land i_{CM}(v_0, v'_1, v'_2). \]  
\[ \{7 + 8, 9 + 10\} \]
Well-definedness for expressions (guards, actions, invariants, etc) needs to be verified. These are verified by means of POs in Event-B \[16\]. 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.

**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 \( evti \) follows from:

\[
\text{REF}_{evti} \equiv I_{i}(v_i) \land J_{i}(v_i, w_i) \land H_{i}(q_i, w_i) \land T_{i}(q_i, w_i, w'_i) \\
\vdash \exists v'_i, G_i(v_i) \land S_i(p_i, v_i, v'_i) \land J_{i}(v'_i, w'_i) \tag{11}
\]

**Theorem 4.** For each composed event \( evt1 \parallel evt2 \), refining abstract event \( evt_0 \) through (gluing) composition invariant in a composed machine, the refinement \( \text{REF PO} \) 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 \[15\] and \[11\]:

\[
\text{INV}_{evti} : \quad I_{i}(w_i) \land H_{i}(q_i, w_i) \land T_{i}(q_i, w_i, w'_i) \vdash i_i(w'_i) \tag{12}
\]

\[
\text{INV}_{evti} : \quad I_{2}(w_2) \land H_{2}(q_2, w_2) \land T_{2}(q_2, w_2, w'_2) \vdash i_2(w'_2) \tag{13}
\]

\[\text{REF}_{evt0 \subseteq (evt1 \parallel evt2)} : \quad I_{0}(v_0) \land I_{1}(w_1) \land I_{2}(w_2) \land J_{CM}(v_0, w_1, w_2) \land H_{1}(q_1, w_1) \land H_{2}(q_2, w_2) \land T_{1}(q_1, w_1, w'_1) \land T_{2}(q_2, w_2, w'_2) \vdash \exists v'_0, G_0(p_0, v_0) \land S_0(p_0, v_0, v'_0) \land I_{1}(w'_1) \land I_{2}(w'_2) \land J_{CM}(v'_0, w'_1, w'_2). \tag{11}\]

**Assume:** \( \text{INV}_{evt1} \tag{12} \) and \( \text{INV}_{evt2} \tag{13} \).

**Prove:** \( \text{REF}_{evt0 \subseteq (evt1 \parallel evt2)} \).

**Proof.** Assume the hypotheses of \( \text{REF}_{evt0 \subseteq (evt1 \parallel evt2)} \). Prove: \( \exists v'_0, G_0(p_0, v_0) \land S_0(p_0, v_0, v'_0) \land I_{1}(w'_1) \land I_{2}(w'_2) \land J_{CM}(v'_0, w'_1, w'_2) \). The proof proceeds as follows:

\[
\exists v'_0, G_0(p_0, v_0) \land S_0(p_0, v_0, v'_0) \\
\land I_{1}(w'_1) \land I_{2}(w'_2) \land J_{CM}(v'_0, w'_1, w'_2) \\
\equiv G_0(p_0, v_0) \land I_{1}(w'_1) \land I_{2}(w'_2) \\
\land \exists v'_0(S_0(p_0, v_0, v'_0) \land J_{CM}(v'_0, w'_1, w'_2)) \quad \{\text{goal; } v_0, w'_1, w'_2 \text{ are free variables}\} \\
\equiv G_0(p_0, v_0) \\
\land \exists v'_0(S_0(p_0, v_0, v'_0) \land J_{CM}(v'_0, w'_1, w'_2)) \quad \{\text{from } (12) \text{ and } (13)\}
\]

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.
4.3 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 [13] using actions systems and CSP. Figure 5 shows abstract component specification $M_1$ composed with other component specification $N_1$, creating a composed model $M_1 \parallel N_1$. $M_1$ is refined by $M_2$ and $N_1$ by $N_2$ respectively. Once we compose specifications $M_1$ and $N_1$, discharge the required composed POs, $M_1$ and $N_1$ can be refined individually while the composition properties are preserved without the need to recompose refinements $M_2$ and $N_2$. We want to formally prove the monotonicity property through refinement POs between composed machines. Therefore if the refinement POs hold between $CM_1$ and $CM_2$ then $CM_1 \preceq CM_2$. Events $evt_{M_1}$ in machine $M_1$ and $evt_{M_2}$ in machine $M_2$ are represented as:

$$evt_{M_1} \equiv \text{ANY } p_M \text{ WHERE } G_M(p_M, v_M) \text{ THEN } S_M(p_M, v_M, v'_M) \text{ END}$$

$$evt_{M_2} \equiv \text{ANY } q_M \text{ WHERE } H_M(q_M, w_M) \text{ THEN } T_M(q_M, w_M, w'_M) \text{ END}$$

The gluing invariant of the refinement between $M_1$ and $M_2$ is expressed as:

$$J_{CM_1}(v_M, w_M, w_{M_1}) = J_{M_1}(v_M, w_M) \land J_{M_2}(v_M, w_{M_2})$$

The refinement PO between $N_2$ and $N_1$ is similar. We refine an abstract event in $CM_1$ by a concrete one in $CM_2$ and verify that the refinement POs for each individual machine hold for the composition. Event $evt_{M_1}$ from machine $M_1$ and event $evt_{N_1}$ from machine $N_1$ are composed, resulting in the abstract composed event $evt_{M_1} \parallel evt_{N_1}$ in $CM_1$ from Fig. 5.

The gluing invariant relating the states of $CM_1$ and $CM_2$ is expressed as the conjunction of the gluing invariants between ($M_1$ and $M_2$) and ($N_1$ and $N_2$):

$$J_{CM}(v_M, v_N, w_M, w_N) = J_{M_1}(v_M, w_M) \land J_{N_1}(v_N, w_N)$$

Fig. 5. Refinement of composed machine $CM_1 \equiv M_1 \parallel N_1$ by $CM_2 \equiv M_2 \parallel N_2$
Theorem 5. 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 $evt_{M_2}$ and abstract composed event $evt_{M_1} \parallel evt_{N_1}$ is expressed as:

$$REF_{evt_{M_1} \parallel evt_{N_1} \subseteq (evt_{M_2} \parallel evt_{N_2})} : I_M(v_M) \land I_N(v_N) \land J_{CM}(v_M, v_N, w_M, w_N)$$
$$\land H_M(q_M, w_M) \land H_N(q_N, w_N)$$
$$\land T_M(q_M, w_M, w_M') \land T_N(q_N, w_N, w_N')$$
$$\land S_M(p_M, v_M, v_M') \land S_N(p_N, v_N, v_N')$$
$$\land J_{CM}(v_M', v_N', w_M', w_N').$$

Assume: $REF_{evt_{M_1} \parallel evt_{N_1}}$ and $REF_{evt_{N_1} \parallel evt_{N_2}}$.

Proof. Assume the hypotheses of $REF_{evt_{M_1} \parallel evt_{N_1} \subseteq (evt_{M_2} \parallel evt_{N_2})}$:

$$J_{CM}(v_M, v_N, w_M, w_N) \equiv J_M(v_M, w_M) \land J_N(v_N, w_N)$$ (expanding $J_{CM}$ from (17))
$$I_M(v_M) \land H_M(q_M, w_M) \land T_M(q_M, w_M, w_M')$$
$$\land S_M(p_M, v_M, v_M') \land S_N(p_N, v_N, v_N')$$
$$\land J_{CM}(v_M', v_N', w_M', w_N').$$

The proof proceeds as follows:

$$\exists v_M', v_N' \cdot G_M(p_M, v_M) \land G_N(p_N, v_N)$$
$$\land S_M(p_M, v_M, v_M') \land S_N(p_N, v_N, v_N')$$
$$\land J_{CM}(v_M', v_N', w_M', w_N').$$

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 $M_1$ and $CM_2$. In this case, the gluing invariant described in (17) does not use neither the variables $(v_N)$ nor the invariants $(I_N)$ from $N_1$. Therefore it can be simplified and rewritten as:

$$J_{CM}(v_M, w_M, w_N) = J_M(v_M, w_M) \land J_N(w_N)$$

Deriving from (19), the goal of $INV_{evt_{M_2} \parallel evt_{N_2}}$ for composition $CM_2$ can be expanded to:

$$j_{CM}(v_M', w_M', w_N') \equiv j_M(v_M', w_M') \land j_N(w_N')$$

where $j_M$ and $j_N$ correspond to each invariant from the set of gluing invariants $J_M$ and $J_N$ respectively.

Theorem 6. An individual event $evt_{M_1}$ in machine $M_1$ is refined by a composed event $evt_{M_2} \parallel evt_{N_2}$ in composed machine $CM_2$. 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 \( \text{evt}_{M2} \parallel \text{evt}_{N2} \). The refinement PO between concrete composed event \( \text{evt}_{M2} \parallel \text{evt}_{N2} \) and abstract non-composed event \( \text{evt}_{M1} \):

\[
\text{REF}_{\text{evt}_{M1} \subseteq (\text{evt}_{M2} \parallel \text{evt}_{N2})} : \quad I_M(v_M) \land J_{CM}(v_M, w_M, w_N) \\
\land H_M(q_M, w_M) \land H_N(q_N, w_N) \\
\land T_M(q_M, w_M, w'_M) \land T_N(q_N, w_N, w'_N) \\
\vdash \exists v'_M \cdot G_M(p_M, v_M) \land S_M(p_M, v_M, v'_M) \\
\land J_{CM}(v'_M, w'_M, w'_N).
\]  

(23)

**Assume:** \( \text{REF}_{\text{evt}_{M1} \subseteq (\text{evt}_{M2} \parallel \text{evt}_{N2})} \) and \( \text{INV}_{\text{evt}_{M2} \parallel \text{evt}_{N2}} \).

**Proof:** \( \text{REF}_{\text{evt}_{M1} \subseteq (\text{evt}_{M2} \parallel \text{evt}_{N2})} \).

Assume the hypotheses of \( \text{REF}_{\text{evt}_{M1} \subseteq (\text{evt}_{M2} \parallel \text{evt}_{N2})} \):

\[
J_{CM}(v_M, w_M, w_N) \equiv J_M(v_M, w_M) \land J_N(w_N) \quad \{ \text{expanding } J_{CM} \text{ from (21)} \}. \\
I_M(v_M) \land H_M(q_M, w_M) \land T_M(q_M, w_M, w'_M) \\
H_N(q_N, w_N) \land T_N(q_N, w_N, w'_N)
\]

(24)

And the hypotheses of \( \text{INV}_{\text{evt}_{M2} \parallel \text{evt}_{N2}} \):

\[
J_{CM}(v_M, w_M, w_N) \equiv J_M(v_M, w_M) \land J_N(w_N) \quad \{ \text{expanding } J_{CM} \text{ from (21)} \}. \\
I_M(v_M) \land H_M(q_M, w_M) \land T_M(q_M, w_M, w'_M) \\
W_2(v'_M, w_M, w_N, q_M, q_N, w'_M, w'_N) \\
H_N(q_N, w_N) \land T_N(q_N, w_N, w'_N)
\]

(25)

(26)

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

\[
\exists v'_M \cdot G_M(p_M, v_M) \land S_M(p_M, v_M, v'_M) \\
\land J_{CM}(v'_M, w'_M, w'_N) \\
\equiv \exists v'_M \cdot G_M(p_M, v_M) \land S_M(p_M, v_M, v'_M) \\
\land J_M(v'_M, w'_M, w'_N) \\
\equiv \exists v'_M \cdot G_M(p_M, v_M) \land S_M(p_M, v_M, v'_M) \\
\land J_N(w'_N) \quad \{ \text{disjoint } v'_M \} \\
\Leftrightarrow \text{REF}_{\text{evt}_{M1} \subseteq \text{evt}_{M2}} \\
\land J_N(w'_N) \quad \{ (16) + (24) \} \\
\Leftrightarrow \text{REF}_{\text{evt}_{M1} \subseteq \text{evt}_{M2}} \\
\land \text{INV}_{\text{evt}_{M2} \parallel \text{evt}_{N2}} \quad \{ (22) + (25) + (26) \}
\]

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

Based on the work developed for composition, its properties and the inverse relation between composition and decomposition, we develop a methodology to partition models in a shared event style. As described in Sect. 3.2 in a shared event decomposition approach, the variables of a system are separated into different sub-components and consequently the rest of the system is decomposed. We present the steps that are required in order to process during decomposition.

Variables: From the modeller’s point of view, the decomposition starts by defining which sub-components are generated. The following step is to define the partition of variables over the sub-components. The rest of the model decomposition (events, parameters, invariants, contexts) is a consequence of the variables allocation as defined below.

Invariants: The decomposition of the invariants depends on the scope of the variables. It is still not very clear which invariants should be retained in the decomposition except the ones related with variable type definition. Further invariants need more study to determine their partition. It seems that they should depend on the input of the user since they might be a constraint of the composed component and not a requirement of the sub-component. When an invariant clause is required but uses variables placed outside the scope of a sub-component, a further refinement of the composed component might be required to make an explicit separation of the variables. An option is to duplicate variables, suggested by Butler [17,18].

Events: The partition of the events depends on the partition of the variables. When the decomposition occurs, parameters are shared between the decomposed events. But the guards referring to that parameter can be different in each decomposed event. The guard of a decomposed event inherits the guard on the composed event according to the variable partition. For example, let us consider event $e_1$:

$$e_1 \equiv \text{WHEN } c = \text{TRUE THEN } a := b \| c := \text{FALSE}$$

where variables $a$ and $b$ are of type $\text{DATA}$ and variable $c$ is a Boolean. This event is enabled when $c$ is TRUE and results in $a$ being assigned the value of $b$ and this event being disabled by assigning $c$ to FALSE. If this event is decomposed such that variable $a$ belongs to one sub-component and variables $b$ and $c$ belong to another, then the action $b := m$ needs to be split. Although the original event does not have parameters, the decomposed events have a new parameter $p$. During the decomposition, that assignment is divided into three steps and a parameter $p$ is introduced:

$$a := b \iff p \in \text{DATA} \land p = b \land a := p$$

Parameter $p$ receives the value of variable $b$. Then the value of $p$ is assigned to variable $a$. The resulting decomposed events are:

$$e'_1 \equiv \text{ANY } p \text{ WHERE } p = b \land c = \text{TRUE THEN } c := \text{FALSE}$$
These corresponds to the value passing of parallel events similar to suggested by Butler [13] for action systems based on CSP: for event $e_1'$, parameter $p$ has a output behaviour as it is written by the value of $b$; in event $e_1''$, parameter $p$ has an input behaviour has the value is read and assigned to variable $a$.

The events in the sub-components resulting from the decomposition maintain the interface of the original events, preserving the parts corresponding to the variables that belongs to each sub-component.

6 File Access Management case study

A distributed system is presented where a system is decomposed into two smaller parts. A specification of a file management system is developed: files containing $DATA$ can be created, read, overwritten, deleted and sent to other users. Each file has an owner. The owners are users with clearance level ranging from 1 to 10 where 10 is the highest level. A super user exists with clearance level 10. Moreover, files have a classification level varying from 1 to 10. Permission is needed in order to read, modify or delete a file. When the permission is granted, the requested action can take place.

Machine FileAccessManagement contains variables user, file, fileData (contains the data of each file) and fileStatus (defines the status of a file operation and can have the states SUCCESS or FAILED). When a file is created or sent, variable fileStatus is updated accordingly to the result of the operation. The status of a file must be reset (in event clearFileStatus) to allow a new operation in the same file. The access management is defined by variables userClearanceLevel, permission, fileClassification and fileOwner. A user can change the clearance of another user as long as the former has a clearance level superior to the latter as described in event modifyUser (guard grd3 in Fig. 6(b)). For all the other operations, permission is required and it is granted by the non-deterministic action in event requestPermission. When a permission is granted, a file can be read, modified, deleted or sent to another user. A file can only be modified by users with a clearance level superior to the file classification (guard grd8 in event overwriteFile). To delete a file, described in event deleteFile, the user must be the owner of the file or be the super user as described by guard grd5.

Our intention is to separate the management of permissions (administrative task) from the modification of the files in the disk (writing, reading tasks). The result are two sub-components, AccessMng and FileMng that deal with different parts of the system. An advantage of this decomposition is that it becomes easier to define specific properties to each part without additional constraints of the other part. For instance, an administrative task of AccessManagement is to have a quota of disk per user which is irrelevant to FileMng. Overwriting a file in the disk is relevant to FileMng but not to AccessMng that deals
Fig. 6. *FileAccessManagement*: variables, invariants (a) and some events (b)
with the users that are allowed to execute this action is not. Therefore we de-
compose \textit{FileAccessManagement} into two sub-components as described in the
next section.

6.1 Decomposition \textit{FileAccessManagement}: \textit{AccessMng} and
\textit{FileMng}

Following the steps suggested in Sect. 5, the variables of \textit{FileAccessManagement}
need to be allocated to sub-components \textit{AccessMng} and \textit{FileMng} as described
in the following table:

<table>
<thead>
<tr>
<th>Variables</th>
<th>FileMng</th>
<th>AccessMng</th>
</tr>
</thead>
<tbody>
<tr>
<td>file, user, fileData, fileStatus</td>
<td>userClearanceLevel, permission, fileOwner, fileClassification</td>
<td></td>
</tr>
</tbody>
</table>

The distribution of events can be seen on the composed machine described in
Fig. 7.


Fig. 7. Composed machine \textit{FileAccessManagement}

Some events are specific to a sub-component: events \textit{modifyUser} and
\textit{requestPermission} belong to \textit{AccessMng} while \textit{clearFileStatus} belongs to
\textit{FileMng}. The majority of the events are decomposed between the two sub-
components and when they are synchronised and occur in parallel, they refine
the original event before the decomposition. The resulting sub-components can
be seen in Figs. 8 and 9. Composition and decomposition are combined when
modelling this system: the decomposition partition the model in sub-components
based on the variables and the composition expresses how the decomposed events
interact. Silva et al. [19] present a decomposition tool that permits the semi-
automatic decomposition in a shared event or shared variable style. Our case
study was run using this tool and we show that we can integrate the shared event decomposition in cooperation with composition (and respective composition tool [20]).

One of the properties of the shared event composition is monotonicity. Therefore sub-components can be further refined independently preserving the verified properties while composed. For instance, machine AccessMng can be refined by defining a more deterministic event requestPermission based on the kind of operation and the user that intends to execute the operation. For machine FileMng, the event sendFile can be further refined by introducing a queue where events would be stored before being processed (create a new file own by the recipient of the file). The independent refinement of the sub-components results in a separation of behaviours and properties that can be verified without the interference of other sub-components.

7 Conclusions

Composition allows the interaction of sub-components. Back [21], Abadi and Lamport [22] studied the interaction of components through shared variable composition. Jones [23] 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 [24] where variables within the same scope cannot have identical names or by views [1] 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, Bellerarde et al [25] suggest a composition by rearranging
Fig. 9. AccessMng (a) and FileMng (b): decomposed events addUser, overwriteFile and deleteFile

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 including a notion of refinement and variable sharing while we use synchronisation and communication in the CSP style. Butler and Walden [20] 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 [8] 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 [27]. Sorge et al [28]
propose a feature composition in Event-B and define composition POs to ensure its consistency. In the feature composition approach, exploration of specifications’ composition with possible variable sharing (similar to the shared variable style) is allowed but no refinement is defined which differs from our work. Nevertheless similar to our work, sub-components POs are reused to avoid re-proving composition POs.

Our Event-B composition and decomposition is based on the close relation between action systems and Event-B plus the correspondence between action systems and CSP [13]. Shared event composition is proved to be monotonic by means of POs. Refinement in a “top-down” style for developing specifications is allowed. Sub-components interact through event parameters by value-passing and can be further refined. We extend Event-B to support shared event composition, allowing combination and reuse of existing sub-components through the introduction of composed machines. Such an approach seems suitable for modelling (distributed) systems. We combine composition and decomposition and suggest a methodology for modelling systems including the verification of properties through the generation of POs and refinement. 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 file access management system is decomposed into two independent parts with a separation of their logics: file management and access management. Possible refinement for each sub-component are suggested to carry on this development. Other case studies have been applying composition with success in particular for distributed systems such as the decomposition of a safe metro system.

References