Shared Event Composition/Decomposition in Event-B*

Renato Silva*** and Michael Butler

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

Abstract. The construction of specifications is often a combination of smaller sub-components. *Composition* and *decomposition* are techniques supporting reuse and allowing formal combination of sub-components through refinement steps. Sub-components can result from a design or architectural goal and a refinement framework should allow them to be further developed, possibly in parallel. We propose the definition of composition and decomposition in the Event-B formalism following a shared event approach where sub-components interact via synchronised shared events and shared states are not allowed. We define the necessary proof obligations to ensure valid compositions and decompositions. We also show that shared event composition preserves refinement proofs, 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

* Part of this research was carried out within the European Commission ICT project 214158 DEPLOY (http://www.deploy-project.eu.

*** R. Silva receives a Doctoral Degree Grant sponsored by Fundação Ciência e Tecnologia (FCT-Portugal).

in Event-B [5] through the interaction of sub-components (modules) , benefiting from their properties and proof obligations (POs). The interesting part of composition involves the interaction of sub-components which usually occurs by means of shared state [6], shared operations [7] or a combination of both (for example, fusion composition [4]). In CSP [8,9] shared actions labels can be synchronised. We take a similar approach in Event-B and we *synchronise events* independently of their labels in a *shared event composition approach*. Properties of the CSP synchronisation such as monotonicity remain valid for the shared event composition. Butler [7] using Action Systems [10] and Classical B [11] defines the parallel composition of action systems including parallel composition with value-passing. We follow this approach to define the shared event composition for Event-B.

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 the respective proofs, decomposition allows team development in parallel over the same model which is very attractive in an industrial environment. Moreover the proof obligations of the original (non-decomposed) model can be reused by the sub-components. The proof obligations to ensure a valid composition are expressed including the possibility to reuse the sub-components properties. We present in more detail the shared event approach applied to composition and decomposition. 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. Guidelines for applying a shared event decomposition are presented illustrated by a case study. The models are developed in Rodin [12], an Event-B toolset [5,13].

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 the shared event approach for composition and decomposition. Composed machines, properties, proof obligations are described in Sect. 4. Decomposition guidelines are presented in Sect. 5. Section 6 illustrates the application of composition and decomposition to a distributed system case study: file access system. Related work is described in Sect. 7. Conclusions and future work are drawn in Sect. 8.

2 Event-B Language

Event-B, inspired by Action Systems, Classical B and Z [14], is a formal modelling method for developing *correct-by-construction* hardware and software systems. An Event-B model is a state transition system where the state corresponds to a set of *variables* v and transitions are represented by *events*. Essential is the formulation of *invariants* $I(v)$: safety conditions to be preserved at all times.

An abstract Event-B specification is divided into a static part called *context* and a dynamic part called *machine* as seen in Fig. 1. A context consists of sets

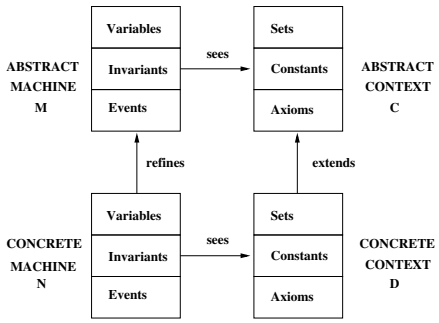

Fig. 1. Machine and context refinement

s (collection of elements or type definitions), constants c and axioms $A(\dots)$[1] of the system. A machine contains the state (global) variables v whose values are assigned in *events*. Events, that can be parameterised by local variables p, occur when their conditions (called *guards* $G(\dots)$) are true and as a result the state variables may be updated by *actions* $S(\dots)$. *Invariants* $I(\dots)$ define the dynamic properties of the specification and POs are generated to verify that these properties are always maintained. The most general form of an event is

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

where event e is expressed by parameters p, guards $G(s,c,p,v)$ and actions $S(s,c,p,v,v')$. When guard $G(s,c,p,v)$ is true then event e is enabled and therefore the action $S(s,c,p,v,v')$ updates the set of variables v to v' (value of v after the assignment).

To facilitate the construction of large-scale models, Event-B advocates the use of *refinement*: the process of gradually adding details to a model. Refinement of a machine consists of refining existing events. An Event-B development is a sequence of models linked by refinement relations. It is said that a concrete model refines an abstract one. Abstract variables v are linked to concrete variables w by a *gluing invariant* $J(v,w)$. POs are generated to ensure that this invariant is preserved in the concrete model. Any behaviour of the concrete model must be *simulated* by some behaviour of the abstract model, with respect to the gluing invariant $J(v,w)$. New events can be added, refinining *skip* which may be declared as convergent, meaning they do not cause divergence. The convergence is proved if each new event decreases a *variant*. The variant must be well-founded and may be an integer or a finite set.

3 Shared Event Approach

The shared event approach is suitable for the development of distributed systems[7]: sub-components interact through synchronised events in parallel. In CSP, syn-

[1] $(\dots)$ refers to the free identifiers in the expression like sets, constants, etc.

chronised input or output channels can exchange messages. In Event-B, the sub-component events can exchange messages via *shared parameters* which is useful for modelling message broadcasting systems. Next we describe how we define a shared event composition in Event-B.

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 semantics of the system. We focus on developments using shared event composition where individual elements' properties are conjoined: *conjunction* of individual invariants, *conjoining* variables and *synchronisation* of events.

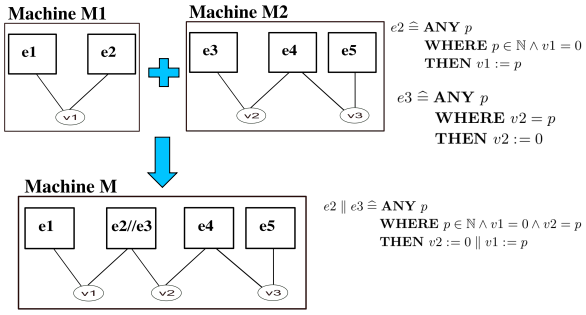

Fig. 2. Shared event composition of *M1* and *M2* (a) resulting in *M* (b)

Consider Fig. 2 where machine *M1* has events *e1* and *e2* using variable *v1*. Moreover machine *M2* has events *e3*, *e4* and *e5* using variables *v2* and *v3*. Events *e2* and *e3* can occur in parallel (independent variables) and can be synchronised. In Fig. 2, machine *M* is the result of the shared event composition of machines *M1* and *M2* where *e2* from machine *M1* and *e3* from machine *M3* are composed: *e2* ‖ *e3*. The interaction of machines *M1* and *M2* through their events results in a *composed event* sharing two independent variables: *v1* and *v2*.

Butler [7] defines a general definition for the parallel composition of action systems with value-passing fusion. Based on that work, we can express a general definition for the parallel composition of generic events e_a and e_b as Def. 1 :

Definition 1. *Composition of events* e_a *and* e_b *with a common parameter* p *results in:*

$$e_a \mathrel{\widehat{=}} \textbf{\textit{ANY}}\ p?, x\ \ \textbf{\textit{WHERE}}\ G(p?, x, m)\ \ \textbf{\textit{THEN}}\ S(p?, x, m)\ \ \textbf{\textit{END}}$$

$$e_b \mathrel{\widehat{=}} \textbf{\textit{ANY}}\ p!, y\ \ \textbf{\textit{WHERE}}\ H(p!, y, n)\ \ \textbf{\textit{THEN}}\ T(p!, y, n)\ \ \textbf{\textit{END}}$$

$$e_a \parallel e_b \mathrel{\widehat{=}} \textbf{\textit{ANY}}\ p!, x, y\ \ \textbf{\textit{WHERE}}\ G(p!, x, m)\ \wedge\ H(p!, y, n)$$
$$\textbf{\textit{THEN}}\ S(p!, x, m) \parallel T(p!, y, n)\ \ \textbf{\textit{END}}$$

where x, y, p are sets of parameters from each of the events e_a and e_b. Event e_a has $p?$ as an input parameter and e_b has $p!$ as an output parameter and the resulting composition is $p!$ itself an output parameter, modelling the passing of the output value from the output parameter to the input parameter. This property can be used to model value-passing systems: e_b sends a value to e_a using the common parameter p. Communication between input type parameters is also possible but not for both output parameters since the output parameters may not be willing to output the same value, leading to a deadlock state. Although it is possible to compose events e_a and e_b even if they share variables, this would lead to a shared variable decomposition which out of the scope of this document since we focus on the shared event decomposition that restricts variable sharing. More information about that kind of composition can be found in [6].

Action systems [10] provide a general description of reactive systems, capable of modelling terminating, aborting and infinitely repeating systems. Event-B is inspired by action systems and can be seen as a realisation of actions systems but using a combination of logic and mathematics. 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 [15,16].

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 [15].

The semantics of the parallel composition of machines M and N corresponds to the set of failure-divergence for each individual machine in parallel. The parallel operator for value-passing action-systems enjoys properties such as monotonicity and associativity [15]. There is a correspondence between action systems and Event-B. Action system is a predicate transformer from a precondition P to post-condition Q with variables v possibly being modified. Event-B events are similar but from a more specific view where the guards correspond to preconditions P, actions R correspond to post-condition Q and the same variables v are possibly modified:

$$[\textbf{ANY } x \textbf{ WHERE } P(x, v) \textbf{ THEN } v :| R(x, v, v') \textbf{ END}]Q$$

An action in action systems is expressed by:

$$(\forall x \cdot P(x, v) \Rightarrow [v :| R(x, v, v')]Q) \Leftrightarrow (\forall x \cdot P(x, v) \Rightarrow (\forall v \cdot R(x, v, v') \Rightarrow Q))$$

Event-B can be seen as a realisation of the generic action system formalism where there is a direct correspondence between Action System actions and Event-B

events. 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. 4.3. An advantage of using Event-B is the tool support available through the Rodin platform where proof obligations are automatically generated.

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_m with variables v_1 to v_m respectively is the conjunction of the individual invariants and the composition invariant $I_{CM}(s, c, v_1, \ldots, v_m)$:*

$$I(M_1 \parallel \cdots \parallel M_m) \mathrel{\widehat{=}} I_1(s, c, v_1) \wedge \cdots \wedge I_m(s, c, v_m) \wedge I_{CM}(s, c, v_1, \ldots, v_m). \quad (1)$$

In Fig. 2, *composed machine M* has as invariant the conjunction of the individual invariants $I(A \parallel B) \mathrel{\widehat{=}} I_A(s, c, v_1) \wedge I_B(s, c, v_2, v_3)$ plus a 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 [17] proposes a shared event decomposition for Event-B inspired by CSP and action systems with event sharing as seen in Fig. 3. We follow that work in our approach.

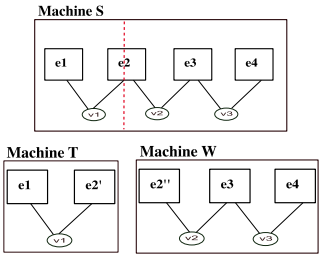

Fig. 3. Shared event decomposition of machine S into T and W sharing *e2*

The decomposition is obtained by selecting which variables from the original model are allocated to which sub-component. Therefore, events using variables allocated to different sub-components (*e2* shares *v1* and *v2*) must be split (described in Sect. 5). The part corresponding to each variable (*e2'* and *e2"*) 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.

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 machines. Machines are composed in parallel including their invariants, variables and events: $CM \mathrel{\widehat{=}} M1 \parallel \cdots \parallel Mm$ as seen in Fig. 4. 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. 2.
- The composed events are defined according to Def. 1.

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
$e_{11} \mathrel{\widehat{=}} M1.e_{11} \parallel \ldots Mm.e_{m1}$
...
$e_{1p} \mathrel{\widehat{=}} M1.e_{1p} \parallel \ldots Mm.e_{m1}\ e_{1p}$
END

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

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 $M1 \dots Mm$ and refining abstract machine $M0$ can be expressed as $M0 \sqsubseteq CM \equiv M0 \sqsubseteq M1 \parallel \cdots \parallel Mm$. 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 such as sets, constants and axioms $(s, c, A(s, c))$ that are part of the static side of a specification, are not considered in the formulas. The POs defined for standard machines are [5]:

- 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)

Invariant Preservation and Gluing Invariant Preservation POs differ in that the first refers to the invariant in the abstract machine while the second refers to invariant relating abstract and concrete variables in a (concrete) refinement machine. 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 explain and define the additional POs necessary to ensure that the composed machine satisfies all the standard POs. We consider that the POs of $M0$, M_1 and M_2 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 $e1 \parallel e2$ is $FIS_{e1 \parallel e2}$.

The Feasibility PO ensures that each non-deterministic action is feasible for a particular event. The goal is to ensure that values exist for variables v' such that the before-after predicate $S(p, s, c, v, v')$ is feasible.

Theorem 2. *The FIS PO of individual events can be reused for proving the feasibility for each composed event and that is enough to verify this property. The feasibility PO for the composed event $e1 \parallel e2$ can be expressed by the feasibility PO of $e1$ (FIS_{e1}) and $e2$ (FIS_{e2}).*

$$FIS_{e1}: \quad I_1(v_1) \land G_1(p_1, v_1) \vdash \exists v_1' \cdot (S_1(p_1, v_1, v_1')) \tag{2}$$

$$FIS_{e2}: \quad I_2(v_2) \land G_2(p_2, v_2) \vdash \exists v_2' \cdot (S_2(p_2, v_2, v_2')) \tag{3}$$

$$FIS_{e1 \parallel e2}: \quad I_{CM}(v_0, v_1, v_2) \land I_1(v_1) \land I_2(v_2) \land G_1(p_1, v_1) \land G_2(p_2, v_2) \vdash \exists v_1', v_2' \cdot (S_1(p_1, v_1, v_1') \land S_2(p_2, v_2, v_2')). \tag{4}$$

Assume: FIS_{e1} *and* FIS_{e2}.
Prove: $FIS_{e1 \| e2}$.

Proof. Assume the hypotheses of $FIS_{e1 \| e2}$.

$$I_{CM}(v_0, v_1, v_2)$$
$$I_1(v_1) \wedge G_1(p_1, v_1) \tag{5}$$
$$I_2(v_2) \wedge G_2(p_2, v_2). \tag{6}$$

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:

$$\begin{aligned}
&\exists v_1', v_2' \cdot (S_1(p_1, v_1, v_1') \wedge S_2(p_2, v_2, v_2')) \\
&\quad \equiv \exists v_1' \cdot (S_1(p_1, v_1, v_1')) \wedge \exists v_2' \cdot (S_2(p_2, v_2, v_2')) && \{\text{disjoint v1 and v2}\} \\
&\Leftarrow (FIS_{e1} \wedge FIS_{e2}). && \{(2)+(5),(3)+(6)\}
\end{aligned}$$

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 $e1 \parallel e2$ is $INV_{e1 \| e2}$.

Theorem 3. *This kind of proof obligation ensures that each invariant is preserved by each event. The goal is each individual invariant from the set of existing invariants. For each invariant i from the set of invariants I in a composed machine, the composition invariant* $I_{CM}(v_0, v_1, v_2)$ *needs to be verified.*

$$INV_{e1}: \quad I_1(v_1) \wedge G_1(p_1, v_1) \wedge S_1(p_1, v_1, v_1') \vdash i_1(v_1') \tag{7}$$
$$INV_{e2}: \quad I_2(v_2) \wedge G_2(p_2, v_2) \wedge S_2(p_2, v_2, v_2') \vdash i_2(v_2') \tag{8}$$
$$\begin{aligned}
INV_{e1 \| e2}: \quad & I_{CM}(v_0, v_1, v_2) \wedge I_1(v_1) \wedge I_2(v_2) \\
& \wedge G_1(p_1, v_1) \wedge G_2(p_2, v_2) \wedge S_1(p_1, v_1, v_1') \wedge S_2(p_2, v_2, v_2') \\
& \vdash i_1(v_1') \wedge i_2(v_2') \wedge i_{CM}(v_0, v_1', v_2')
\end{aligned}$$

Assume: INV_{e1} *and* INV_{e2}.
Prove: $INV_{e1 \| e2}$.

Proof. Assume the hypotheses of $INV_{e1 \| e2}$.

$$I_{CM}(v_0, v_1, v_2)$$
$$I_1(v_1) \wedge G_1(p_1, v_1) \wedge S_1(p_1, v_1, v_1') \tag{9}$$
$$I_2(v_2) \wedge G_2(p_2, v_2) \wedge S_2(p_2, v_2, v_2') \tag{10}$$

Prove: $i_1(v_1') \wedge i_2(v_2') \wedge i_{CM}(v_0, v_1', v_2')$. The proof proceeds as follows:

$$\begin{aligned}
&i_1(v_1') \wedge i_2(v_2') \wedge i_{CM}(v_0, v_1', v_2') \\
&\Leftarrow INV_{e1} \wedge INV_{e2} \wedge i_{CM}(v_0, v_1', v_2'). && \{(7)+(9),(8)+(10)\}
\end{aligned}$$

Well-definedness for expressions (guards, actions, invariants, etc) needs to be verified. These are verified by means of POs in Event-B [18]. 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 e_0 is refined by composed machine CM defined by machines M_1 with variables w_1, invariant $I_1(w_1)$, event e_1 and M_2 (w_2 ; $I_2(w_2)$; e_2) and composition invariant $J_{CM}(v_0, w_1, w_2)$. The composed event $e1 \parallel e2$ refines the abstract event e_0. The refinement PO results from the verification of the invariant preservation $J_M(v_0, w_i)$, the verification of guard strengthening for $G_0(p_0, v_0)$ and simulation $S_0(p_0, v_0, v_0')$ for each concrete event. A general refinement PO (REF_{ei}) for a machine M refining event ei follows from:

$$\begin{aligned} REF_{ei} \mathrel{\widehat{=}} & I_i(v_i) \land J_i(v_i, w_i) \land H_i(q_i, w_i) \land T_i(q_i, w_i, w_i') \\ & \vdash \exists v_i' \cdot G_i(v_i) \land S_i(p_i, v_i, v_i') \land J_i(v_i', w_i') \end{aligned} \tag{11}$$

Theorem 4. *For each composed event $e1 \parallel e2$, refining abstract event $e0$ 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* (11)*:*

$$INV_{e1}: \quad I_1(w_1) \land H_1(q_1, w_1) \land T_1(q_1, w_1, w_1') \vdash i_1(w_1') \tag{12}$$

$$INV_{e2}: \quad I_2(w_2) \land H_2(q_2, w_2) \land T_2(q_2, w_2, w_2') \vdash i_2(w_2') \tag{13}$$

$$\begin{aligned} REF_{e0 \sqsubseteq (e1 \parallel e2)}: \quad & I_0(v_0) \land I_1(w_1) \land I_2(w_2) \land J_{CM}(v_0, w_1, w_2) \\ & \land H_1(q_1, w_1) \land H_2(q_2, w_2) \land T_1(q_1, w_1, w_1') \land T_2(q_2, w_2, w_2') \\ & \vdash \exists v_0' \cdot G_0(p_0, v_0) \land S_0(p_0, v_0, v_0') \land I_1(w_1') \land I_2(w_2') \land J_{CM}(v_0', w_1', w_2') \end{aligned}$$

Assume: INV_{e1} (12) *and* INV_{e2} (13).
Prove: $REF_{e0 \sqsubseteq (e1 \parallel e2)}$.

Proof. Assume the hypotheses of $REF_{e0 \sqsubseteq (e1 \parallel e2)}$. Prove: $\exists v_0' \cdot G_0(p_0, v_0) \land S_0(p_0, v_0, v_0') \land I_1(w_1') \land I_2(w_2') \land J_{CM}(v_0', w_1', w_2')$. The proof proceeds as follows:

$$\begin{aligned} & \exists v_0' \cdot G_0(p_0, v_0) \land S_0(p_0, v_0, v_0') \\ & \land I_1(w_1') \land I_2(w_2') \land J_{CM}(v_0', w_1', w_2') \\ \equiv\ & G_0(p_0, v_0) \land I_1(w_1') \land I_2(w_2') \\ & \land \exists v_0' \cdot (S_0(p_0, v_0, v_0') \land J_{CM}(v_0', w_1', w_2')) \qquad \{\land \text{ goal; } v_0, w_1', w_2' \text{ are free variables}\} \\ \equiv\ & G_0(p_0, v_0) \\ & \land \exists v_0' \cdot (S_0(p_0, v_0, v_0') \land J_{CM}(v_0', w_1', w_2')) \qquad \{\text{from (12) and (13)}\} \end{aligned}$$

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.

4.3 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 that this property holds as it happens for actions systems and CSP described by Butler [15]. Figure 5

shows abstract component specification $M1$ composed with other component specification $N1$, creating a composed model $M1 \parallel N1$. $M1$ is refined by $M2$ and $N1$ by $N2$ respectively. Once we compose 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 property

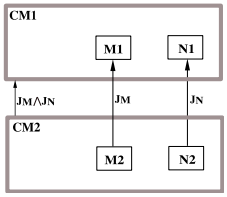

Fig. 5. Refinement of composed machine $CM1 \mathrel{\widehat{=}} M1 \parallel N1$ by $CM2 \mathrel{\widehat{=}} M2 \parallel N2$

through refinement POs between composed machines. Therefore if the refinement POs hold between $CM1$ and $CM2$ then $CM1 \sqsubseteq CM2$. Events e_{M1} in machine $M1$ and e_{M2} in machine $M2$ are represented as:

$$e_{M1} \mathrel{\widehat{=}} \textbf{ANY } p_M \textbf{ WHERE } G_M(p_M, v_M) \textbf{THEN } S_M(p_M, v_M, v'_M) \textbf{ END} \tag{14}$$

$$e_{M2} \mathrel{\widehat{=}} \textbf{ANY } q_M \textbf{ WHERE } H_M(q_M, w_M) \textbf{THEN } T_M(q_M, w_M, w'_M) \textbf{ END} \tag{15}$$

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 e_{M2} refining abstract event e_{M1}.

$$\begin{aligned} REF_{e_{M1} \sqsubseteq e_{M2}} : \quad & I_M(v_M) \wedge J_M(v_M, w_M) \wedge G_M(p_M, v_M) \wedge H_M(q_M, w_M) \\ & \wedge S_M(p_M, v_M, v'_M) \wedge T_M(q_M, w_M, w'_M) \\ & \vdash \exists v'_M \cdot G_M(p_M, v_M) \wedge S_M(p_M, v_M, v'_M) \wedge J_M(v'_M, w'_M). \end{aligned} \tag{16}$$

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 e_{M1} from machine $M1$ and event e_{N1} from machine $N1$ are composed, resulting in the abstract composed event $e_{M1} \parallel e_{N1}$ in $CM1$ from Fig. 5. 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 $N2$):

$$J_{CM}(v_M, v_N, w_M, w_N) = J_M(v_M, w_M) \wedge J_N(v_N, w_N) \tag{17}$$

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 $e_{M2} \parallel e_{N2}$ and abstract*

composed event $e_{M1} \parallel e_{N1}$ *is expressed as:*

$$\begin{aligned} REF_{(e_{M1}\parallel e_{N1})\sqsubseteq(e_{M2}\parallel e_{N2})}: \quad & I_M(v_M) \wedge I_N(v_N) \wedge J_{CM}(v_M, v_N, w_M, w_N) \\ & \wedge H_M(q_M, w_M) \wedge H_N(q_N, w_N) \\ & \wedge T_M(q_M, w_M, w'_M) \wedge T_N(q_N, w_N, w'_N) \\ & \vdash \exists v'_M, v'_N \cdot G_M(p_M, v_M) \wedge G_N(p_N, v_N) \\ & \wedge S_M(p_M, v_M, v'_M) \wedge S_N(p_N, v_N, v'_N) \\ & \wedge J_{CM}(v'_M, v'_N, w'_M, w'_N). \end{aligned} \tag{18}$$

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

Proof. Assume the hypotheses of $REF_{(e_{M1}\parallel e_{N1})\sqsubseteq(e_{M2}\parallel e_{N2})}$.

$$J_{CM}(v_M, v_N, w_M, w_N) \equiv J_M(v_M, w_M) \wedge J_N(v_N, w_N) \quad \{\text{expanding } J_{CM} \text{ from (17)}\}$$

$$I_M(v_M) \wedge H_M(q_M, w_M) \wedge T_M(q_M, w_M, w'_M) \tag{19}$$

$$I_N(v_N) \wedge H_N(q_N, w_N) \wedge T_N(q_N, w_N, w'_N) \tag{20}$$

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:

$$\begin{aligned} & \exists v'_M, v'_N \cdot G_M(p_M, v_M) \wedge G_N(p_N, v_N) \\ & \quad \wedge S_M(p_M, v_M, v'_M) \wedge S_N(p_N, v_N, v'_N) \\ & \quad \wedge J_M(v'_M, w'_M) \wedge J_N(v'_N, w'_N) && \{\text{expanding } J_{CM} \text{ from (17)}\} \\ & \equiv \exists v'_M \cdot G_M(v_M) \wedge S_M(p_M, v_M, v'_M) \wedge J_M(v'_M, w'_M) \\ & \quad \wedge \exists v'_N \cdot G_N(v_N) \wedge S_N(p_N, v_N, v'_N) \wedge J_N(v'_N, w'_N) && \{\text{disjoint } v'_M, v'_N\} \\ & \Leftarrow REF_{e_{M1}\sqsubseteq e_{M2}} \wedge REF_{e_{N1}\sqsubseteq e_{N2}} && \{(16)+(19),(16)+(20)\} \end{aligned}$$

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 (17) does not use neither the variables (v_N) neither the invariants(I_N) neither events (e_{N1}) from $N1$. Therefore it can be simplified and rewritten as:

$$J_{CM}(v_M, w_M, w_N) = J_M(v_M, w_M) \wedge J_N(w_N) \tag{21}$$

Deriving from (21), the goal of $INV_{e_{M2}\parallel e_{N2}}$ can be expanded to:

$$j_{CM}(v'_M, w'_M, w'_N) \equiv j_M(v'_M, w'_M) \wedge j_N(w'_N) \tag{22}$$

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* e_{M1} *in machine* $M1$ *is refined by a composed event* $e_{M2} \parallel e_{N2}$ *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* $e_{M2} \parallel e_{N2}$. *The refinement*

PO between concrete composed event $e_{M2} \parallel e_{N2}$ and abstract non-composed event e_{M1}:

$$\begin{aligned} REF_{e_{M1} \sqsubseteq (e_{M2} \parallel e_{N2})} : \quad & I_M(v_M) \land J_{CM}(v_M, w_M, w_N) \land H_M(q_M, w_M) \\ & \land H_N(q_N, w_N) \land T_M(q_M, w_M, w'_M) \land T_N(q_N, w_N, w'_N) \\ & \vdash \exists v'_M \cdot G_M(p_M, v_M) \land S_M(p_M, v_M, v'_M) \land J_{CM}(v'_M, w'_M, w'_N) \end{aligned} \tag{23}$$

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

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

$$\begin{aligned} & J_{CM}(v_M, w_M, w_N) \equiv J_M(v_M, w_M) \land J_N(w_N) \quad \{\text{expanding } J_{CM} \text{ from (21)}\}. \\ & I_M(v_M) \land H_M(q_M, w_M) \land T_M(q_M, w_M, w'_M) \\ & H_N(q_N, w_N) \land T_N(q_N, w_N, w'_N) \end{aligned} \tag{24}$$

And the hypotheses of $INV_{e_{M2} \parallel e_{N2}}$:

$$\begin{aligned} & J_{CM}(v_M, w_M, w_N) \equiv J_M(v_M, w_M) \land J_N(w_N) \quad \{\text{expanding } J_{CM} \text{ from (21)}\} \\ & I_M(v_M) \land H_M(q_M, w_M) \land T_M(q_M, w_M, w'_M) \\ & W_2(v'_M, w_M, w_N, q_M, q_N, w'_M, w'_N) \qquad (25) \\ & H_N(q_N, w_N) \land T_N(q_N, w_N, w'_N) \qquad (26) \end{aligned}$$

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:

$$\begin{aligned} & \exists v'_M \cdot G_M(p_M, v_M) \land S_M(p_M, v_M, v'_M) \\ & \land J_{CM}(v'_M, v'_N, w'_M, w'_N) \\ \equiv\ & \exists v'_M \cdot G_M(p_M, v_M) \land S_M(p_M, v_M, v'_M) \\ & \land J_M(v'_M, w'_M) \land J_N(w'_N) && \{\text{expanding } J_{CM} \text{ from (21)}\} \\ \equiv\ & \exists v'_M \cdot G_M(p_M, v_M) \land S_M(p_M, v_M, v'_M) \land J_M(v'_M, w'_M) \\ & \land J_N(w'_N) && \{\text{disjoint } v'_M\} \\ \Leftarrow\ & REF_{e_{M1} \sqsubseteq e_{M2}} \land J_N(w'_N) && \{(16)+(24)\} \\ \Leftarrow\ & REF_{e_{M1} \sqsubseteq e_{M2}} \land INV_{e_{M2} \parallel e_{N2}} && \{(22)+(25)+(26)\} \end{aligned}$$

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, for 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. As a restriction of the shared event approach, no variable sharing is allowed. We present the required steps to process a decomposition, possible problems and how to tackle them.

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 variable allocation as defined below.

Invariants: The decomposition of the invariants depends on the scope of the variables. Therefore the minimal set of invariants must include the variable type definitions. And these are the required invariants for a valid refinement. Additional ones depend on the user, as they may be useful in later refinements or to help in reusing the sub-components. When an invariant clause is demanded and 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.

Events: The partition of the events depends on the partition of the variables. When the decomposition occurs, parameters are shared between the decomposed events. The guard of a decomposed event inherits the guard on the composed event according to the variable partition. For example, let us consider event $e1$:

$$e1 \mathrel{\widehat{=}} \textbf{WHEN } c = \text{TRUE } \textbf{THEN } a := b \ \parallel c := \text{FALSE}$$

where variables a and b are of type $DATA$ 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 sub-component *M1* and variables b and c belong to *M2*, then action $a := b$ needs to be split. This assignment needs to be rewritten in a way that these variables are not part of the same expression. A solution is to refine this event in a way that the guards and actions do not refer to variables allocated to different sub-components. Before the decomposition, we refine event *e1* by adding parameter p:

$$\begin{aligned} e1 \mathrel{\widehat{=}} \textbf{ANY } p \textbf{ WHEN } & c = \text{TRUE} \wedge p \in DATA \wedge p = b \\ \textbf{THEN } & a := p \ \parallel c := \text{FALSE} \end{aligned}$$

Parameter p receives the value of variable b. Then the value of p is assigned to variable a. The parameter p is shared between the sub-components and whereas variable a is within the scope of *M1* only containing the guard $p \in DATA$ and the action $a := p$ ($e1'$), the guard $p = b$ is added to *M2* ($e1''$):

$$\begin{aligned} e1' &\mathrel{\widehat{=}} \textbf{ANY } p \textbf{ WHERE } p \in DATA \textbf{ THEN } a := p \\ e1'' &\mathrel{\widehat{=}} \textbf{ANY } p \textbf{ WHERE } p \in DATA \wedge p = b \wedge c = \text{TRUE } \textbf{THEN } c := \text{FALSE} \end{aligned}$$

These corresponds to the value passing of parallel events similar to suggested by Butler [15] for action systems based on CSP: for event $e1''$, parameter p has a output behaviour as it is written by the value of b; in event $e1'$, parameter p has an input behaviour as its 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). 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 aspects of the system. An advantage of this separation is to more easily 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 with the users that are allowed to execute this action. Therefore we decompose *FileAccessManagement* into two sub-components as described in the next section.

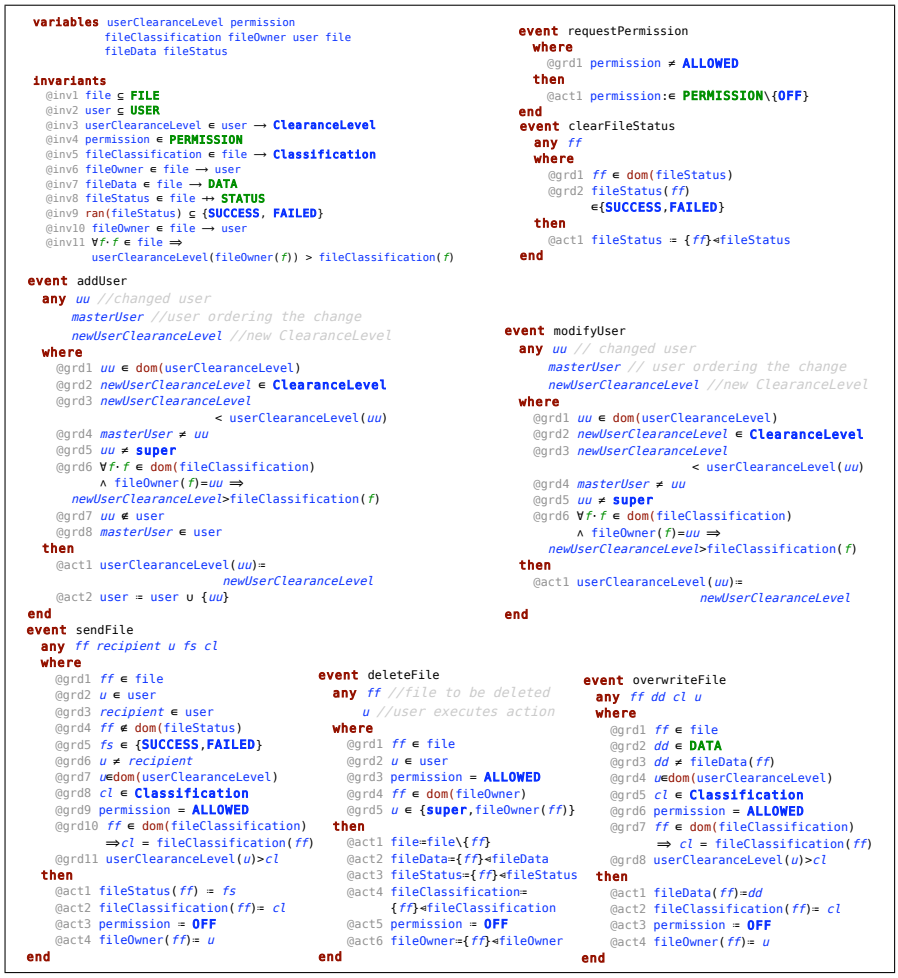

Fig. 6. *FileAccessManagement*: variables, invariants and some events

6.1 Decomposition: *AccessMng* and *FileMng*

Following the steps suggested in Sect. 5, the variables of *FileAccessManagement* are allocated to *AccessMng* and *FileMng* as described in the following table:

	FileMng	AccessMng
Variables	file,user, fileData,fileStatus	userClearanceLevel,permission, fileOwner,fileClassification

The distribution of events can be seen on the composed machine described in Fig. 7. Some events are specific to a sub-component: events *modifyUser* and *requestPermission* belong to *AccessMng* while *clearFileStatus* belongs to *FileMng*; the other events are shared. In Fig. 8, the invariants include theorems defining the

COMPOSED MACHINE *FileAccessManagement*
INCLUDES
AccessMng, *FileMng*
EVENTS
addUser $\widehat{=}$ *AccessMng.addUser* $\parallel$ *FileMng.addUser*
modifyUser $\widehat{=}$ *AccessMng.modifyUser*
createFile$\widehat{=}$ *AccessMng.createFile* $\parallel$ *FileMng.createFile*
readFile $\widehat{=}$ *AccessMng.readFile* $\parallel$ *FileMng.readFile*
overwriteFile $\widehat{=}$ *AccessMng.overwriteFile* $\parallel$ *FileMng.overwriteFile*
deleteFile $\widehat{=}$ *AccessMng.deleteFile* $\parallel$ *FileMng.deleteFile*
sendFile $\widehat{=}$ *AccessMng.sendFile* $\parallel$ *FileMng.sendFile*
requestPermission $\widehat{=}$ *AccessMng.requestPermission*
clearFileStatus $\widehat{=}$ *FileMng.clearFileStatus*

Fig. 7. Composed machine *FileAccessManagement*

variable types as suggested in Sect. 5. Moreover for Fig. 8(b), invariants relating variables for the same sub-component are automatically included. Figure 9 shows the decomposed events *overwriteFile* where parameters *ff*, *dd* and *cl* are shared (value passing from *AccessMng* to *FileMng*). Also the actions are split according to the user's variable selecting (cf. Table above): $fileOwner$, $fileClassification$ and *permission* belong to *AccessMng* while $fileData$ belongs to *FileMng*.

```
machine AccessMng sees
User_C0 AccessManagement_C0 FileManagement_C0

variables userClearanceLevel permission
          fileClassification fileOwner

invariants
  theorem @typing_userClearanceLevel
           userClearanceLevel ∈ ℙ(USER × ℤ)
  theorem @typing_fileOwner
           fileOwner ∈ ℙ(FILE × USER)
  theorem @typing_permission
           permission ∈ PERMISSION
  theorem @typing_fileClassification
           fileClassification ∈ ℙ(FILE × ℤ)
```

(a)

```
machine FileMng sees
User_C0 AccessManagement_C0 FileManagement_C0

variables file user fileData fileStatus

invariants
  theorem @typing_fileStatus fileStatus∈ℙ(FILE×STATUS)
  theorem @typing_file file ∈ ℙ(FILE)
  theorem @typing_user user ∈ ℙ(USER)
  theorem @typing_fileData fileData ∈ ℙ(FILE × DATA)
  @FileAccessMng_inv1 file ⊆ FILE
  @FileAccessMng_inv2 user ⊆ USER
  @FileAccessMng_inv7 fileData ∈ file ⟶ DATA
  @FileAccessMng_inv8 fileStatus ∈ file ⇸ STATUS
  @FileAccessMng_inv9 ran(fileStatus) ⊆ {SUCCESS, FAILED}
```

(b)

Fig. 8. *AccessMng* (a) and *FileMng* (b): variables and invariants

Composition and decomposition are combined: the decomposition partitions the model in sub-components based on the variables and the composition expresses the events' interaction. The extensibility of Rodin, allows new functionalities to be added to the Event-B language. *Silva et al* [19] developed a semi-automatic decomposition tool for shared event or shared variable. A composition tool [20] is also available in the Rodin platform. We use both tools: *FileAccessManagement* is decomposed using the decomposition tool and the composition tool shows the event splitting. In a shared event decomposition, the user does not control the event splitting since they are a consequence of the variable allocation (selected by the user). Therefore the composition view gives an additional insight of the entire process, complementing the decomposition view.

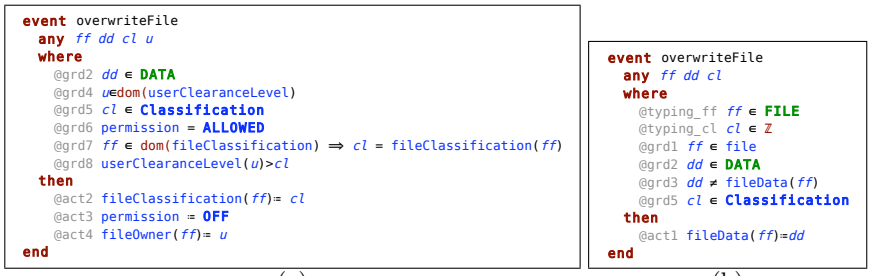

Fig. 9. Decomposed events *overwriteFile* for *AccessMng* (a) and *FileMng* (b)

As we proved in Sect. 4.3, shared event composition is monotonic and consequently sub-components can be further refined independently preserving the verified properties while composed. For instance, *AccessMng* can be refined by defining a more deterministic event *requestPermission* based on the kind of operation and user. For *FileMng*, event *sendFile* can be further refined by introducing a processing queue where events can be stored. The advance of independent refinement of sub-components is a separation of behaviours and properties verifiable without the interference of other sub-components.

7 Related Work

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 [14] 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, Bellegarde et at [24] 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 including a notion of refinement and variable sharing while we use synchronisation and communication in the CSP style. Butler and Walden [25] 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 [6] 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 [26].

8 Conclusions

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 as described in Sect. 3.1. Composition POs are defined to ensure valid composed machines and refinements. These can be simplified when machine POs are reused. We prove that shared event composition is monotonic by means of POs and "top-down" refinement is allowed. Sub-components interact through event parameters by value-passing. Event-B is extended to support *shared event composition*, allowing combination and reuse of existing sub-components through the introduction of *composed machines*. We do not address the step corresponding to the translation of the composition to an implementation. This study needs to be carried out in the future. Using a case study, composition, decomposition and refinement are combined, suggesting a methodology for modelling distributed systems and verifying properties through the generation of POs. A file access management system is decomposed into two independent parts with a separation of their logics: file and access management and possible refinements are suggested. Other case studies have been applying (de)composition with success such as the decomposition of a safety metro system [2].

References

1. Jackson, D.: Structuring Z specifications with views. ACM Trans. Softw. Eng. Methodol. **4**(4) (1995) 365–389
2. Zave, P., Jackson, M.: Conjunction as Composition. ACM Trans. Softw. Eng. Methodol. **2**(4) (1993) 379–411
3. Jones, C.B.: Wanted: a compositional approach to concurrency. In: Programming methodology. Springer-Verlag New York, Inc., New York, NY, USA (2003) 5–15
4. 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
5. Abrial, J.R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press (2010)
6. Abrial, J.R., Hallerstede, S.: Refinement, Decomposition, and Instantiation of Discrete Models: Application to Event-B. Fundam. Inf. **77**(1-2) (2007) 1–28
7. 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

[2] This case study is available online at `http://eprints.ecs.soton.ac.uk/22195/`

8. Hoare, C.A.R.: Communicating Sequential Processes. Prentice Hall International Series in Computer Science (1985)
9. Morgan, C.: Of wp and CSP. In: Beauty is our business: a birthday salute to Edsger W. Dijkstra. Springer-Verlag New York, Inc., New York, NY, USA (1990) 319–326
10. Ralph-Johan R. Back, Kurki-Suonio, R.: Decentralization of Process Nets with Centralized Control. In: PODC '83: Proceedings of the second annual ACM symposium on Principles of distributed computing, New York, NY, USA, ACM (1983) 131–142
11. Abrial, J.R.: The B-Book: Assigning programs to meanings. Cambridge University Press (1996)
12. Rodin: RODIN project Homepage. `http://rodin.cs.ncl.ac.uk` (September 2008) Online; accessed 27-July-2010.
13. Abrial, J.R., Butler, M.J., Hallerstede, S., Voisin, L.: An Open Extensible Tool Environment for Event-B. In: ICFEM. (2006) 588–605
14. Spivey, J.M.: The Z Notation: a Reference Manual. Prentice-Hall, Inc. (1989)
15. Butler, M.J.: A CSP Approach to Action Systems. PhD thesis, Oxford University (1992)
16. Butler, M.: Stepwise Refinement of Communicating Systems. Science of Computer Programming **27**(2) (September 1996) 139–173
17. Butler, M.: Synchronisation-Based Decomposition for Event-B. In: RODIN Deliverable D19 Intermediate report on methodology. (2006) 47–57
18. Abrial, J.R., Butler, M., Hallerstede, S., Hoang, T.S., Mehta, F., Voisin, L.: Rodin: An Open Toolset for Modelling and Reasoning in Event-B. International Journal on Software Tools for Technology Transfer (STTT) (April 2010)
19. Silva, R., Pascal, C., Hoang, T.S., Butler, M.: Decomposition Tool for Event-B. Software: Practice and Experience **41**(2) (February 2011) 199–208
20. Silva, R., Butler, M.: Parallel Composition Using Event-B. `http://wiki.event-b.org/index.php/Parallel_Composition_using_Event-B` (July 2009) Online; accessed 27-July-2010.
21. 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
22. 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
23. 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
24. 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
25. Butler, M., Waldén, M.: Distributed System Development in B. Technical Report TUCS-TR-53, Turku Centre for Computer Science (14, 1996)
26. Hoang, T., Abrial, J.R.: Event-B Decomposition for Parallel Programs. Abstract State Machines, Alloy, B and Z (2010) 319–333