

Event Model Decomposition

by J.-R. Abrial

24th of December 2001

Version 3 (to be completed)

Event Model Decomposition

1 Introduction

The process of developing an event model by successive refinement steps usually starts with a very few events (sometimes even a single event) dealing with a very few state variables. It usually ends up with many events and many variables. This is because one of the most important mechanism of this approach consists of introducing *new* events, supposed to refine *skip*, during refinements steps. The refinement mechanism is also used at the same time to significantly enlarge the number of state variables. The new events, let us recall, are manifestation of the refinement of the *time grain* within which we might, more and more accurately, observe and analyse our dynamic system.

At some point, we might have so many events and so many state variables that the refinement process might become quite heavy. And we may also figure out that the refinement steps we are trying to undertake are not involving any more the totality of our system (as was the case at the beginning of the development): only a few variables and events are concerned, the others only playing a passive, *but noisy*, rôle.

The idea of *model decomposition* is thus clearly very attractive: it consists of cutting an heavy event system into *smaller pieces* which can be handled more comfortably than the whole. More precisely, each piece should be refinable *independantly* of the others. But, of course, the constraint that must be satisfied by this decomposition is that such independently refined pieces could always (in principle) be easily recomposed. This process should then result in a system which could have been obtained directly without the decomposition, which thus appears to be just a kind of “divide-and-conquer” artefact.

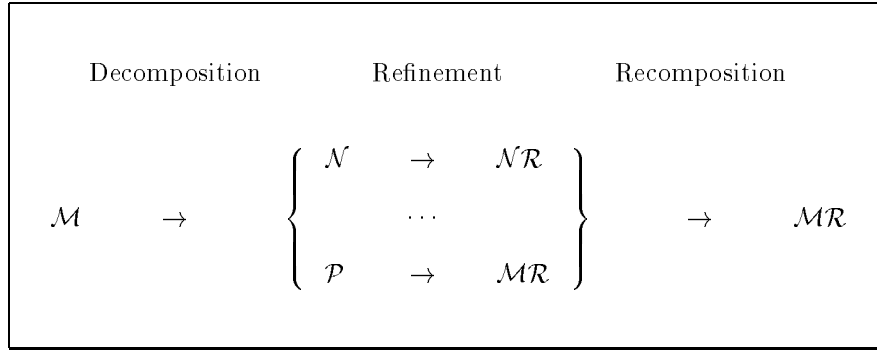
This paper contains the feasibility study of such a mechanism. After proposing an informal definition of decomposition in the next section, we outline the outcome and constraints of this approach (sections 3). We then present its main difficulty (section 4) and propose a solution to it (section 5), which nevertheless presents a certain limitation (as shown in section 5.2). In section 6, we propose a semi-formal proof of our previous solution. In section 7, a short example shows the mechanism at work. In section 8, we present another more elaborate example showing how the main limitation of this approach, as described in section 5.2, can be overcome.

2 Informal Definition

Decomposing an event model \mathcal{M} is a complex process which can be defined in the following way:

1. \mathcal{M} is first “splitted” (in a certain way to be defined) into several models, say $\mathcal{N}, \dots, \mathcal{P}$.
2. These new models are then be refined *independently of each other* yielding $\mathcal{NR}, \dots, \mathcal{PR}$.
3. These refined models are eventually “put together” (also in a certain way to be defined) to form yet another model \mathcal{MR} .
4. The recomposed model \mathcal{MR} is *guaranteed* to be a refinement of \mathcal{M} .

This process is illustrated in the following diagram:



It is important to notice here that point 3 above (the recomposition) will never be performed in practice. One has only to figure out that it *can be done* under certain circumstances and that the refinement condition stated in point 4 is then satisfied.

The events of the component models $\mathcal{N}, \dots, \mathcal{P}$ of the decomposition are supposed to form a *partition* of the events of the initial model \mathcal{M} . Concerning the way the variables of \mathcal{M} are splitted among $\mathcal{N}, \dots, \mathcal{P}$, the problem is not so simple as we shall see below in section 4.

3 Outcome and Constraints of Decomposition

This decomposition process may play three important practical methodological rôles:

1. It is certainly easier (less proofs) to refine (possibly several times) $\mathcal{N}, \dots, \mathcal{P}$ independently of each other rather than together.
2. This process of decomposition is “monotonic” in that refinements of $\mathcal{N}, \dots, \mathcal{P}$ can be further decomposed in the same way, and so on.
3. The models $\mathcal{N}, \dots, \mathcal{P}$ could already possess (off the shelf) some refinements that can then be reused in several projects.

In buiding this mechanism, we shall observe several “constraints”:

1. We must define a process that is *totally robust* mathematically speaking.
2. We do not want to modify in any way the mathematical definition and concept of refinement.
3. We figure out that the process of *decomposition* of an event model is distinct (although close) from that of *importation* as described in “classical B”.

4 The Main Difficulty: Variable Splitting

Suppose that we have a certain model, \mathcal{M} , with four events $m1, m2, m3$ et $m4$. We would like to decompose \mathcal{M} into two separate models: (1) \mathcal{N} dealing with events $m1$ and $m2$, and (2) \mathcal{P} dealing with events $m3$ and $m4$. We are interested in doing this decomposition because we “know” that there are some nice refinements that can be performed on $m1$ and $m2$ (possibly adding some new events) and independently on $m3$ and $m4$ in the same way.

But in doing this *event splitting* we must also perform a certain *variable splitting*. Suppose that we have three variables $v1, v2$ and $v3$ in \mathcal{M} . Like the events, the variables must be splitted too. For instance, we might put $v1$ and $v2$ with \mathcal{N} (because $m1$ and $m2$ are supposedly working

with them and not with $v3$), and thus $v3$ goes, quite naturally, with \mathcal{P} . But the difficulty here is that $m3$ and $m4$, which certainly work with $v3$, *might also work with $v2$* , so that, besides $v3$, \mathcal{P} certainly also requires $v2$ to deal correctly with $m3$ and $m4$.

The problem seems unsovable since apparently there will always be some *shared variables*. As a matter of fact, we have the very strong impression that the splitting of the events will always conflict that of the variables. Suppose it is not the case. In other words, suppose that, in our example, $m1$ and $m2$ only work with $v1$ and $v2$, while $m4$ only does with $v3$. Clearly then, \mathcal{M} is made of two completely separated groups of events ($m1$ and $m2$ in one hand, and $m3$ and $m4$ in the other) which do not communicate in any way with each other. In this case, \mathcal{M} is obviously made of two distinct models, which could have been handled separately to begin with.

So, in all interesting cases, the question of shared variables, $v2$ in our example, is unavoidable. How are we going to solve this difficulty?

5 The Solution: Variable Sharing

5.1 Shared Variables Replication

We have no choice: the shared variables must clearly be *replicated* in the various components of our decomposition. Notice that the shared variables in question can be modified by any of the components: we do not want to make any “specialization” of the components, some of them being only allowed to “read”, and some other to “write” these variables. We know that it is not possible in general.

The new difficulty that arise immediately at this point concerns the problem of refinement. In principle, each component can freely data-refine its state space. So that the *same* replicated variable could, in principle, be refined in one way in one component and differently in another: this is obviously not acceptable.

5.2 A Notion of External Variable

The price to pay in order to solve this difficulty is to give the replicated variables a *special status* in the components where they stay. Let us call this status: *external*. An external variable has a simple limitation: it must always be present in the state space of any refinement of the component. In other words, an external variable *cannot be data-refined*. We shall see in section 8 how this apparent limitation can be overcome.

5.3 A Notion of External Event

But this is not sufficient. Suppose that in a certain component an external variable is only read, not written. The trouble with that external variable is that it has suddenly become a constant in that component, which is certainly not what we want.

What we need thus in each component, is a number of extra events *simulating* the way our external variables are handled in the initial model. Such events are called *external events*. Each of them “mimicks”, using the external variables only, an event of the initial model that modifies the external variables in question. The reader has understood: “mimick” simply means “is an abstraction of”. Of course such external events cannot be refined in their component.

Notice that there is a distinction to be made between an external variable and an external event. An external variable is external in all sub-models where it can be found, whereas an external event always has a non-external counterpart elsewhere. An event, however, can be external in several sub-models (such a case is shown in the example studied in section 8).

5.4 Final Recomposition

The recomposition of the initial model by means of refinements of the various components is now extremely simple. We put together all the variables of the individual components (“dereplicating” the various shared variables) and we simply throw away all the external events of each component.

It remains now for us to prove that the recomposed model is indeed a refinement of the initial one. Notice again that this recomposition is usually not done explicitly. It is just something that could be done, and which must then yield a refinement of the initial model.

6 Proof

We shall make the proof on our example. It can then be easily generalized. Suppose that the usage of the variables $v1$, $v2$ and $v3$ in model \mathcal{M} is as follows:

variable	events
$v1$	m1, m2
$v2$	m2, m3
$v3$	m3, m4

event	variables
m1	$v1$
m2	$v1, v2$
m3	$v2, v3$
m4	$v3$

Model \mathcal{N} uses variable $v1$ as a normal variable and variable $v2$ as an *external* variable. It has events m1 and m2 plus an extra *external event* m3a (for m3 abstracted) dealing with variable $v2$ only. Event m3a is supposed to be refined by event m3. Similarly, model \mathcal{P} uses variable $v3$ as a normal variable and variable $v2$ as an *external* variable. It has events m3 and m4 plus an extra *external event* m2a (for m2 abstracted) dealing with variable $v2$ only. Event m2a is supposed to be refined by event m2. This can be summarized in the following table:

model	variables	events	ext. vbls.	ext. evts.
\mathcal{N}	$v1$	m1, m2	$v2$	m3a
\mathcal{P}	$v3$	m3, m4	$v2$	m2a

It can easily be seen that models \mathcal{N} and \mathcal{P} are both refined by model \mathcal{M} . This is so because events m1 and m2 of \mathcal{N} are clearly refined by events m1 and m2 of \mathcal{M} ; event m3a of \mathcal{N} is refined *by construction* by event m3 of \mathcal{M} ; finally event m4 of \mathcal{M} clearly refines skip in \mathcal{N} since it deals with variables $v3$ which does not exist in \mathcal{N} . And similarly for \mathcal{P} .

Suppose that we now refine \mathcal{N} to \mathcal{NR} . Model \mathcal{NR} has variables $w1$ and $v2$ together with the gluing invariant $J(v1, w1, v2)$. Model \mathcal{NR} has events m1r and m2r which are supposed to be

refinements of $m1$ and $m2$ respectively, and also event $m3a$, which is not refined by convention. Similarly, we refine \mathcal{P} to \mathcal{PR} . Model \mathcal{PR} has variables $w3$ and $v2$ together with the gluing invariant $K(v3, w3, v2)$. Model \mathcal{PR} has events $m3r$ and $m4r$ which are supposed to be refinements of $m3$ and $m4$ respectively, and also event $m2a$, which is not refined by convention. Notice that both gluing invariants J and K depend on the “external” variable $v2$. All this can be summarized in the following table:

model	variables	events	ext. vbls.	ext. evts.	gl. invt.
\mathcal{NR}	$w1$	$m1r, m2r$	$v2$	$m3a$	$J(v1, w1, v2)$
\mathcal{PR}	$w3$	$m3r, m4r$	$v2$	$m2a$	$K(v3, w3, v2)$

Let us now construct a certain model \mathcal{MR} as follows. The state of \mathcal{MR} is made of: (1) the state of \mathcal{NR} , where we remove $v2$ and the part of the invariant of \mathcal{NR} dealing with it, and (2) the state of \mathcal{PR} , where we remove $v2$ and the part of the invariant of \mathcal{PR} dealing with it, and finally (3) the variable $v2$ itself together with the invariants where it appears in both \mathcal{NR} and \mathcal{PR} . The events of \mathcal{MR} are $m1r, m2r, m3r$ and $m4r$. Notice that $m2a$ and $m3a$ have been thrown away. This can be summarized in the following table:

model	variables	events	gluing invariant
\mathcal{MR}	$w1, v2, w3$	$m1r, m2r, m3r, m4r$	$J(v1, w1, v2) \wedge K(v3, w3, v2)$

Clearly \mathcal{NR} and \mathcal{PR} are refined by \mathcal{MR} , but it is not obvious that \mathcal{M} is refined by \mathcal{MR} , this is precisely what we have to prove. The situation is depicted in the following diagram, where the arrows indicates a refinement relationship:

$$\begin{array}{ccccc}
 \mathcal{N} & \rightarrow & \mathcal{M} & \leftarrow & \mathcal{P} \\
 \downarrow & & ? & & \downarrow \\
 \mathcal{NR} & \rightarrow & \mathcal{MR} & \leftarrow & \mathcal{PR}
 \end{array}$$

In what follows we shall prove that, provided $m1r$ and $m2r$ are refinements of $m1$ and $m2$ respectively in \mathcal{NR} , then they also are refinement of $m1$ and $m2$ in \mathcal{MR} . Similar proofs can be conducted for the other events of \mathcal{MR} . Next are the before-after predicates of the concerned events:

event	before-after predicate
m1	$M_1(v1, v1')$
m2	$M_2(v1, v2, v1', v2')$
m1r	$M_{1r}(w1, w1')$
m2r	$M_{2r}(w1, v2, w1', v2')$

The correct refinement condition of m1 into m1r within \mathcal{NR} is the following:

$$\begin{array}{l}
J(v1, w1, v2) \wedge M_{1r}(w1, w1') \\
\Rightarrow \\
\exists v1' \cdot (M_1(v1, v1') \wedge J(v1', w1', v2))
\end{array}$$

Under this hypothesis, the following correct refinement condition of m1 into m1r within \mathcal{MR} clearly holds:

$$\begin{array}{l}
J(v1, w1, v2) \wedge K(v3, w3, v2) \wedge M_{1r}(w1, w1') \\
\Rightarrow \\
\exists v1' \cdot (M_1(v1, v1') \wedge J(v1', w1', v2) \wedge K(v3, w3, v2))
\end{array}$$

As can be seen, condition $K(v3, w3, v2)$ can be extracted from the existential quantification in the consequent of this implication (this is because it does not contain any reference to the quantified variable $v1'$). It is then easily discharged because it is already present in the antecedent of the implication.

The situation is a bit different in the case of the event m2: this is because this event modifies variable $v2$. Next is the correct refinement condition of m2 into m2r within \mathcal{NR} :

$$\begin{array}{l}
J(v1, w1, v2) \wedge M_{2r}(w1, v2, w1', v2') \\
\Rightarrow \\
\exists v1' \cdot (M_2(v1, v2, v1', v2') \wedge J(v1', w1', v2'))
\end{array}$$

The correct refinement condition of m1 into m1r within \mathcal{MR} is:

$$\begin{array}{l}
J(v1, w1, v2) \wedge K(v3, w3, v2) \wedge M_{2r}(w1, v2, w1', v2') \\
\Rightarrow \\
\exists v1' \cdot (M_2(v1, v2, v1', v2') \wedge J(v1', w1', v2') \wedge K(v3, w3, v2'))
\end{array}$$

As above with $K(v3, w3, v2)$, the condition $K(v3, w3, v2')$ can be extracted from the existential quantification in the consequent of this implication. But this time the situation is different from the previous one as we still have the condition $K(v3, w3, v2)$ in the antecedent, not $K(v3, w3, v2')$, so that the proof is not trivial. Again, the presence of $v2'$ in the consequent is due to the fact that $v2$ is modified by $m2$ and $m2r$.

Fortunately, we have not yet exploited the fact that event $m2$ of \mathcal{M} and \mathcal{N} is in fact abstracted within \mathcal{P} by event $m2a$. Suppose that the before-after predicate of $m2a$ is $M_{2a}(v2, v2')$. The corresponding refinement condition of $m2a$ by $m2$ is:

$$\exists v1'. M_2(v1, v2, v1', v2') \Rightarrow M_{2a}(v2, v2')$$

Finally, we also have not exploited the fact that event $m2a$ of \mathcal{P} is *not refined* in \mathcal{PR} . As a consequence, it is clearly part of the “normal” refinement conditions of \mathcal{PR} to prove that condition $K(v3, w3, v2)$ is left invariant under $m2a$. This yields:

$$K(v3, w3, v2) \wedge M_{2a}(v2, v2') \Rightarrow K(v3, w3, v2')$$

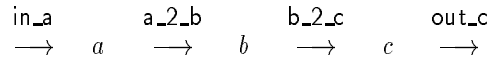
Putting all these conditions together yields the following to prove, which holds “trivially”:

$$\begin{aligned} & K(v3, w3, v2) \wedge M_{2a}(v2, v2') \Rightarrow K(v3, w3, v2') \\ & \exists v1'. M_2(v1, v2, v1', v2') \Rightarrow M_{2a}(v2, v2') \\ & J(v1, w1, v2) \wedge M_{2r}(w1, v2, w1', v2') \Rightarrow \exists v1'. (M_2(v1, v2, v1', v2') \wedge J(v1', w1', v2')) \\ & J(v1, w1, v2) \\ & K(v3, w3, v2) \\ & M_{2r}(w1, v2, w1', v2') \\ & \Rightarrow \\ & \exists v1'. (M_2(v1, v2, v1', v2') \wedge J(v1', w1', v2') \wedge K(v3, w3, v2')) \end{aligned}$$

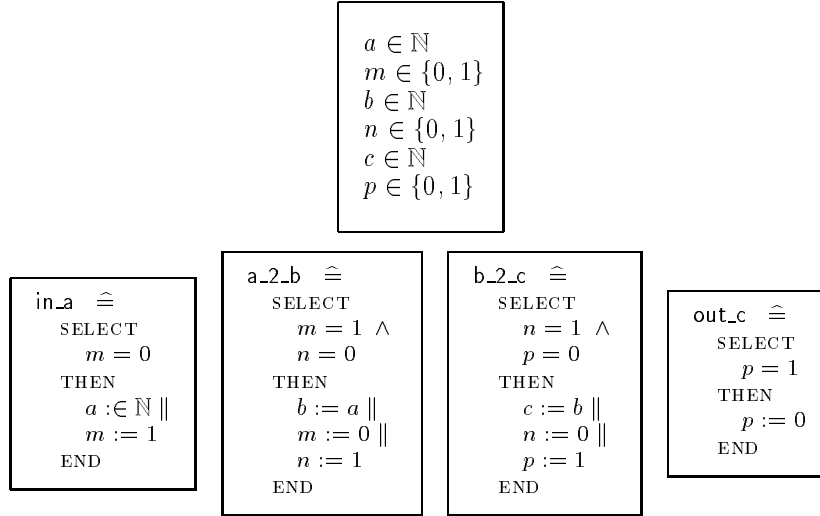
7 A Short Example

7.1 The Initial Model

In an initial model \mathcal{M} , we have three variables a , b , and c handling some natural number values. These variables are “controlled” by three boolean variables m , n , and p respectively. We also have four events called `in_a`, `a_2_b`, `b_2_c`, and `out_c`. These events respectively insert a “new” value in a , move values from a to b , and from b to c , and finally remove values from c as shown in the following diagram:

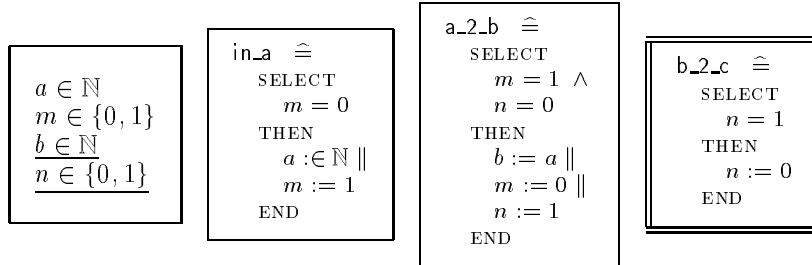


The moving of the values should respect some constraints handled by the control variables. When the control variable, say m , is equal to 1, this means that the value of a is “fresh” so that it can be sent to b . This, however, can only be performed provided the value of b is “old”, meaning that it has already been sent to c : this fact is recorded by the value of n , the boolean variable controlling b , being equal to 0. Next are the typing of the variables and the definition of the various events:

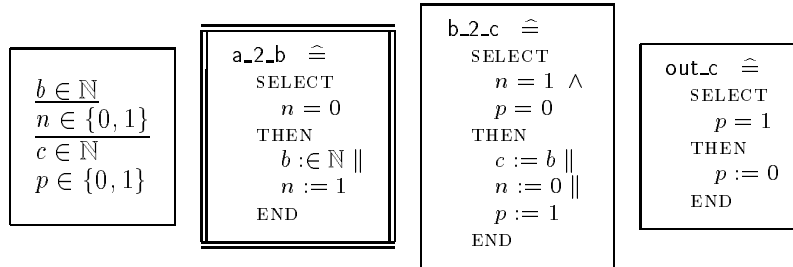


7.2 The Decomposition

We now decompose \mathcal{M} into two models \mathcal{N} and \mathcal{P} . The model \mathcal{N} has proper events in_a and a_2_b , which are thus exact copies of events bearing the same names in model \mathcal{M} , and the “external” event b_2_c (we have double-boxed it below), which must then be an abstraction of the event bearing the same name in the initial model \mathcal{M} and in model \mathcal{P} below. It is also made of the four variables a , m , b , and n , the last two being “external” (we have underlined them below).

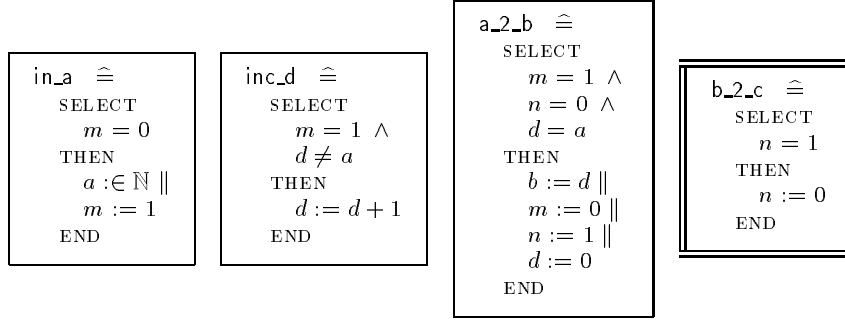


Next is our second decomposed model \mathcal{P} . It is organized symmetrically to \mathcal{N} . Likewise, it is easy to prove that the external event a_2_b is an abstraction of the event with the same name in \mathcal{M} and \mathcal{N} .



7.3 Refinements

We now refine \mathcal{N} to \mathcal{NR} . The refinement consists of adding a new variable d . The value of the variable a is gradually moved to d by incrementing d . For this, we introduce a new event `inc_d` (refining `skip`). Next are the events of \mathcal{NR} (notice that events `in_a` and `b_2_c` are not modified):

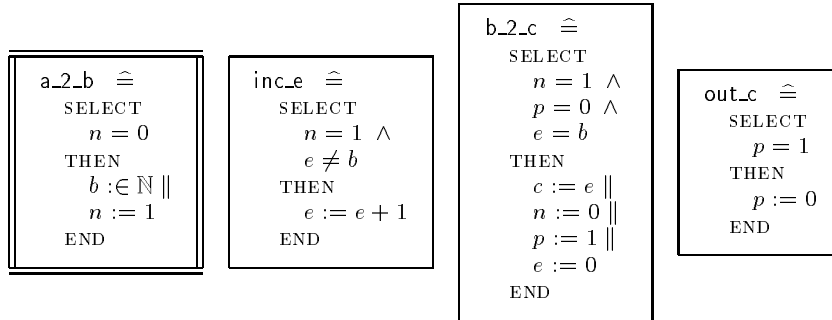


We have three invariants stating that: (1) d is not greater than a (so that the new event `inc_d` cannot take control for ever), (2) d is null when m is, and (3) b is equal to a when m is null and n is 1. Formally:

$$\begin{aligned}
& d \leq a \\
& m = 0 \Rightarrow d = 0 \\
& m = 0 \wedge n = 1 \Rightarrow b = a
\end{aligned}$$

We notice that the third invariant is the only one involving the external variables n and b . It is clearly maintained by the external event `b_2_c`. As a consequence, this invariant is a fortiori maintained by the “genuine” event `b_2_c` in \mathcal{P} and also by its refinement in \mathcal{PR} below.

We now refine similarly \mathcal{P} to \mathcal{PR} . The refinement consists of adding a new variable e . The value of the variable b is gradually moved to e by incrementing e . For this, we introduce a new event `inc_e`. Next are the events of \mathcal{PR} (notice that events `a_2_b` and `out_c` are not modified):



We have three invariants stating that: (1) e is not greater than b (so that event `inc_e` cannot take control for ever), (2) e is null when n is, and (3) c is equal to b when n is null and p is 1. Formally:

$$\begin{aligned}
& e \leq b \\
& n = 0 \Rightarrow e = 0 \\
& n = 0 \wedge p = 1 \Rightarrow c = b
\end{aligned}$$

We notice that, this time, all three invariants involve the external variables n and b . It is easy to prove that these invariants are all maintained by the external event a_2_b . As a consequence, these invariants are a fortiori maintained by the “genuine” event a_2_b in \mathcal{N} and also by its refinement in \mathcal{NR} above.

8 Another Example

In this example, our intention is to show how the variables that appears to be the most natural candidates to be “shared” in a certain decomposition can indeed be refined. The idea is to encapsulate them in separate sub-systems, where they will not be shared any more, and can thus be freely refined. This encapsulation is made possible by the presence of some *simpler* shared variables playing the rôles of *concrete buffers*.

8.1 The Initial model

Our initial model has three events: `elaborate_question`, `elaborate_response`, and `acquire_response`. Each event is associated with a corresponding process. The first one generates a, so-called, “question” (a new one each time), which is sent to the second process by means of a certain $q_channel$. The second process reads the $q_channel$ and produce a “response” to the selected question. It then sends the couple “question-response” to the third process through the $r_channel$. The third process eventually reads this channel and stores the selected couple.

Let $QUESTION$ be the set of questions and $RESPONSE$ be the set of response. The variables of this system, namely $old_questions$, $q_channel$, $r_channel$, and $old_responses$, satisfy the following invariant:

$$\begin{aligned}
 &old_questions \subseteq QUESTION \\
 &q_channel \subseteq old_questions \\
 &r_channel \in (old_questions - q_channel) \rightarrow RESPONSE \\
 &old_responses \in (old_questions - q_channel) \rightarrow RESPONSE \\
 &dom(old_responses) \cap dom(r_channel) = \emptyset
 \end{aligned}$$

Here is our first event, which non-deterministically elaborate a fresh question:

```

elaborate_question  $\hat{=}$ 
  ANY q THEN
    q  $\in$  QUESTIONS - old_questions
  THEN
    q_channel := q_channel  $\cup$  {q} ||
    old_questions := old_questions  $\cup$  {q}
  END

```

Our second event non-deterministically elaborates a response to a selected question in the $q_channel$ (non-deterministically, because we are not interested here in the exact way a relevant response is given to a certain question):

```

elaborate_response ≐
  ANY q, r THEN
    q ∈ q_channel ∧
    r ∈ RESPONSE
  THEN
    q_channel := q_channel - {q} ||
    r_channel := r_channel ∪ {q ↦ r}
  END

```

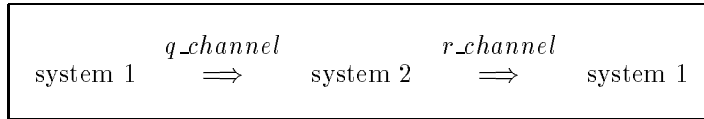
The third event stores questions and corresponding responses:

```

acquire_response ≐
  ANY q, r THEN
    q ∈ QUESTION ∧
    r ∈ RESPONSE ∧
    q ↦ r ∈ r_channel
  THEN
    r_channel := r_channel - {q ↦ r} ||
    old_responses := old_responses ∪ {q ↦ r}
  END

```

Our idea is now to, quite naturally, cut this event system into three pieces. The two channels are “clearly” the natural candidates to be the shared variables between these three sub-systems, as indicated by the following diagram:



But we want to have the possibility to later in depth *refine the channels*. This means that they cannot be shared as indicated. The idea is to isolate them in separate sub-systems and have them communicating with the other sub-systems by means of “buffers”, which will not be refined and thus will be shared. The decomposition will be realized in several steps. First we cut the initial systems into three systems, the channels being parts of the first and last one. Before this, however, we first have to refine our initial system.

8.2 Refinement of the Initial System

In this refinement, we introduce a buffer, named *question_1*, associated with the *q_channel*, and two buffers, named *question_2* and *response_1*, associated with the *r_channel*. The channels themselves are data-refined in a straightforward fashion to *q_channel_1* and *r_channel_1*. We also introduce two booleans variables, *reading_q* and *writing_r*, whose rôle is to properly synchronize the events. We finally have two new events (refining skip) called *reading_question* and *writing_response*. Here are the typing and invariants of our new variables:

```

q_channel_1 ⊆ QUESTION
r_channel_1 ∈ QUESTION → RESPONSE
question_1 ∈ QUESTION
question_2 ∈ QUESTION
response_1 ∈ RESPONSE
reading_q ∈ {0, 1}
writing_r ∈ {0, 1}
reading_q = 0 ⇒ q_channel = q_channel_1
reading_q = 1 ⇒ q_channel = q_channel_1 ∪ {question_1}
reading_q = 1 ⇒ question_1 ∉ q_channel_1
writing_r = 0 ⇒ r_channel = r_channel_1
writing_r = 1 ⇒ r_channel = r_channel_1 ∪ {question_2 ↦ response_1}
writing_r = 1 ⇒ question_2 ∉ dom(r_channel_1)

```

As can be seen, the abstract *q_channel* is the same as its refinement *q_channel_1* when *reading_q* is 0 (meaning that the response to the last selected question has just been elaborated). When *reading_q* is 1 (meaning that there is a fresh question to be answered in the buffer *question_1*) then the abstract *q_channel* is not yet updated whereas the concrete *q_channel_1* already is (the selected question has been removed from *q_channel_1* and put in the buffer *question_1*). Moreover, in that case, the question in the buffer is not any more within the concrete channel. A similar gluing invariant exists for the *r_channel*.

Next are the refined events. In the next one *q_channel* is just replaced by its concrete counterpart *q_channel_1*

```

elaborate_question ≐
  ANY q THEN
    q ∈ QUESTIONS − old_questions
  THEN
    q_channel_1 := q_channel_1 ∪ {q} ||
    old_questions := old_questions ∪ {q}
  END

```

Here is a new event (refining *skip*). It reads the *q_channel_1* and puts the selected question in the buffer *question_1*.

```

reading_question ≐
  ANY q THEN
    q ∈ q_channel_1 ∧
    reading_q = 0
  THEN
    reading_q := 1 ||
    q_channel_1 := q_channel_1 − {q} ||
    question_1 := q
  END

```

The concrete event *elaborate_response* now works with the buffers rather than directly with the channels as was the case in the abstraction.

```

elaborate_response  $\hat{=}$ 
SELECT
  reading_q = 1   $\wedge$ 
  writing_r = 0
THEN
  reading_q := 0 ||
  writing_r := 1 ||
  question_2 := question_1 ||
  response_1  $\in$  RESPONSE
END

```

Next is another new event writing the response from the buffers into the corresponding concrete channel.

```

writing_response  $\hat{=}$ 
SELECT
  writing_r = 1
THEN
  writing_r := 0 ||
  r_channel_1 := r_channel_1  $\cup$  {question_2  $\mapsto$  response_1}
END

```

Our last event is almost the same as its abstraction, the concrete response channel replacing the abstract one.

```

acquire_response  $\hat{=}$ 
ANY q, r THEN
  q  $\in$  QUESTION  $\wedge$ 
  r  $\in$  RESPONSE  $\wedge$ 
  q  $\mapsto$  r  $\in$  r_channel_1
THEN
  r_channel_1 := r_channel_1 - {q  $\mapsto$  r} ||
  old_responses := old_responses  $\cup$  {q  $\mapsto$  r}
END

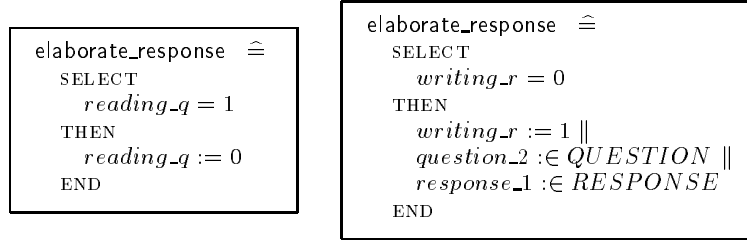
```

8.3 First Decomposition

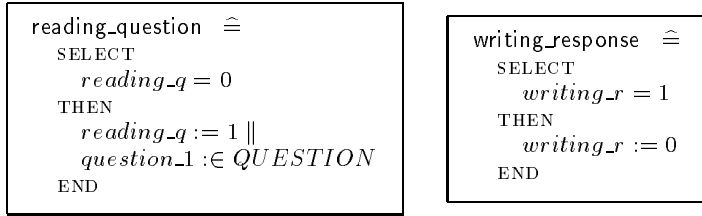
We are now ready to proceed with our first splitting. It is done according to the following table:

System	Variables	Ext. Vbls.	Events	Ext. Evt.
System_1	<i>old_questions</i> <i>q_channel_1</i>	<i>reading_q</i> <i>question_1</i>	elaborate_question reading_question	elaborate_response
System_2		<i>reading_q</i> <i>question_1</i> <i>writing_r</i> <i>question_2</i> <i>response_1</i>	elaborate_response	reading_question writing_response
System_3	<i>r_channel_1</i> <i>old_responses</i>	<i>writing_r</i> <i>question_2</i> <i>response_1</i>	writing_response acquire_response	elaborate_response

Notice that the event `elaborate_response` is external in two systems. Next are its shape in `System_1` and `System_3` respectively;



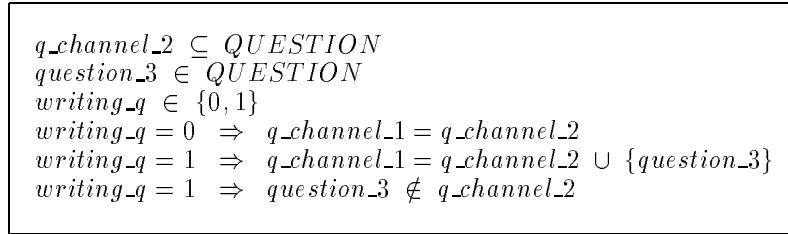
The events `reading_question` and `writing_response` are external in `System_2` as indicated below:



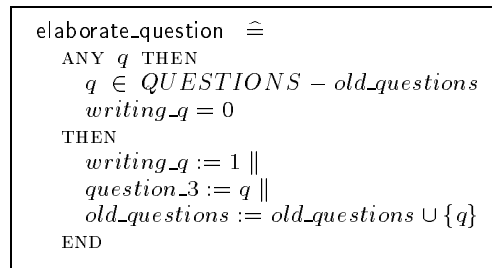
8.4 Refinements of the first and last sub-systems

Our next step consists of splitting our first and last sub-systems. Before doing that, however, we must refine them. This is done in very much the same way as we did previously.

For refining `System_1`, we introduce a buffer named `question_3` and a boolean variable called `writing_q`. The channel `q_channel_1` is refined to `q_channel_2`. We also introduce a new event named `writing_question`. Here are the corresponding typing and invariants:



Next are the refinements of the events. Notice that we cannot refine the event `elaborate_response` because it is external in `System_1`. The new version of the event `elaborate_question` is as follows:



We now have the new event `writing_question`

```

writing_question ≐
SELECT
  writing_q = 1
THEN
  writing_q := 0 ||
  q_channel_2 := q_channel_2 ∪ {question_3}
END

```

Next is the refinement of event `reading_question`:

```

reading_question ≐
ANY q WHERE
  q ∈ q_channel_2 ∧
  reading_q = 0
THEN
  reading_q := 1 ||
  q_channel_2 := q_channel_2 - {q} ||
  question_1 := q
END

```

For refining `System_2`, we introduce two buffers named `question_4` and `response_2`, and a boolean variable called `reading_r`. The channel `r_channel_1` is refined to `r_channel_2`. We also introduce a new event named `writing_question`. Here are the corresponding typing and invariants:

```

r_channel_2 ⊆ QUESTION ⇔ RESPONSE
question_4 ∈ QUESTION
response_2 ∈ RESPONSE
reading_r ∈ {0, 1}
reading_r = 0 ⇒ r_channel_1 = r_channel_2
writing_q = 1 ⇒ q_channel_1 = q_channel_2 ∪ {question_4 ⇔ response_2}
writing_q = 1 ⇒ question_4 ⇔ response_2 ∉ r_channel_2

```

Next are the refinements of the events. Notice again that we cannot refine the event `elaborate_response` because it is external in `System_2`. The new version of the event `writing_response` is as follows:

```

writing_response ≐
SELECT
  writing_r = 1
THEN
  writing_r := 0 ||
  r_channel_2 := r_channel_2 ∪ {question_2 ⇔ response_1}
END

```

Here is the new event `reading_response`

```

reading_response ≐
ANY q, r WHERE
  q ∈ QUESTION ∧ r ∈ RESPONSE ∧
  q ⇔ r ∈ r_channel_2 ∧
  reading_r = 0
THEN
  reading_r := 1 ||
  question_4 := q ||
  response_2 := r ||
  r_channel_2 := r_channel_2 - {q ⇔ r}
END

```

Here is the refinement of event `acquire_response`.

```

acquire_response  $\hat{=}$ 
  SELECT
    reading_r = 1
  THEN
    reading_r := 0 ||
    old_responses := old_responses  $\cup$  {question_4  $\mapsto$  response_2}
  END

```

8.5 Second Decomposition

We now proceed with our second decomposition. It consists of splitting `System_1` and `System_3` in the following way (although not concerned by this splitting, we have nevertheless mentioned `System_2` in this table in order to visualize a complete figure):

System	Variables	Ext. Vbls.	Events	Ext. Evts.
System_11	<i>old_questions</i>	<i>writing_q</i> <i>question_3</i>	elaborate_question	writing_question
System_12	<i>q_channel_2</i>	<i>writing_q</i> <i>question_3</i> <i>reading_q</i> <i>question_1</i>	writing_question reading_question	elaborate_question elaborate_response
System_2		<i>reading_q</i> <i>question_1</i> <i>writing_r</i> <i>question_2</i> <i>response_1</i>	elaborate_response	reading_question writing_response
System_31	<i>r_channel_2</i>	<i>writing_r</i> <i>question_2</i> <i>response_1</i> <i>reading_r</i> <i>question_4</i> <i>response_2</i>	writing_response reading_response	elaborate_response acquire_response
System_32	<i>old_responses</i>	<i>reading_r</i> <i>question_4</i> <i>response_2</i>	acquire_response	reading_response

As can be seen, the variables `q_channel_2` and `r_channel_2` are now properly encapsulated as genuine variables in separate sub-systems. From now on, they can therefore be freely refined, which was our main goal.

To be completed

9 Discussion and comparisons

To be completed

10 Conclusion

To be completed

Acknowledgements: To be completed

REFERENCES: To be completed