Recasting Monolithic Event-B Specifications into Modular $\mathcal{EVT}$

Marie Farrell

Department of Computer Science
Maynooth University

1 Cars on a Bridge

This is a formalisation in $\mathcal{EVT}$ using specification-building operators of the cars on a bridge example outlined in Chapter 2 of Abrial's book [?]. For all steps in this development we provide the Event-B machine and relevant contexts to show what we are trying to emulate and we also provide an associated $\mathcal{EVT}$ specification that is more modular. The approaches to modularisation that we have taken are by no means the only ones available. There are many ways to write the same specification. The Event-B specifications below were all obtained directly (as a .zip file) from the Event-B wiki page which contains specifications corresponding to all of the systems described in the book.

1.1 The Abstract Machine, M0

Figure 1 contains an Event-B specification corresponding to the first abstract machine in this development. It describes the behaviour of cars entering and leaving the mainland. This abstract model is comprised of a context specifying a natural number constant and a basic machine description. We form the corresponding modular $\mathcal{EVT}$ specification that is contained in Figure 2 as follows:

Lines 1–5: This is a $\mathcal{CASL}$ specification that describes the context from Figure 1. The constant d is represented as an operation of the appropriate type and axioms are included as predicates.

Lines 6–13: This is a specification that describes the data contained in the machine and also contains the `Initialisation` event. Separating the data component of the machine is a development strategy that we have used throughout this document but it is not the only approach that could have been taken. It also would have been possible to parametrise the specification of the machine by it's data. This would be a more elegant approach, however, it is also more complex for the unfamiliar reader so we have opted for a strictly specification-building approach in this case study.

Lines 14–19 and 20–25: These contain specifications that describe the behaviour of leaving and entering, we have chosen to include these as separate specifications so that they will be easy to reuse later.

Lines 26–29: This is the full machine specification that takes a copy of the leave and enter specifications using with to rename the events appropriately.

```
CONTEXT cd
  CONSTANTS
    d
  AXIOMS
      axm1: d ∈ ℕ
      axm2: d > 0
END
```

```
MACHINE m0
  SEES cd
  VARIABLES
    n
  INVARIANTS
     inv1: n ∈ ℕ
     inv2: n ≤ d
     inv3: n < 0 ∨ n < d
  EVENTS
    Initialisation
      then
         act1: n := 0
    Event ML_out ≙ordinary
      when
         grd1: n < d
      then
         act1: n := n + 1
     Event ML_in ≙ordinary
       when
          grd1: n > 0
       then
          act1: n := n − 1
END
```

Fig. 1. Event-B abstract machine with context.

Even though this is quite a small and simple Event-B model, it is easy to see that the corresponding $\mathcal{EVT}$ specification is, by far, more modular. This will be further evidenced by the reuse of the leave and enter specifications in what follows.

1.2 The First Refinement, M1

In the first refinement step, as can be seen in Figure 3, new events are added to describe cars entering and leaving the island. New variables are added to record the number of cars on the island, cars on the mainland and those on the bridge. The events ML_in and ML_out are also refined to utilise these new variables. Figure 4 contains an $\mathcal{EVT}$ specification corresponding to this Event-B machine specification. It is comprised of the following modular specifications:

Lines 1–11: As was the approach taken with the abstract machine, this specification describes the data to be used and also a specification of the `Initialisation` event. Note that we have omitted those invariants that are labelled as theorems because they should be derived from the specification. It is possible to include them using the %implied annotation that is made available in Hets but we have omitted them for simplicity.

Lines 12–21: This specification describes the island events by combining two copies of the Enter specification that was constructed as part of the abstract $\mathcal{EVT}$ specification in Figure 2. We use with to provide appropriate renaming of events and variables.

Lines 22–30: This describes the refined mainland events. Note that where multiple events with the same name appear in a specification, they are assumed to be merged. This is the basic $\mathcal{CASL}$ notion of "same name, same thing".

```
spec CD over 𝒞𝒜𝒮ℒ =
  sort ℕ
  ops d : ℕ
  . d > 0
end

spec DataM0 over ℰ𝒱𝒯 =
  CD with ρ
  then
    sort ℕ
    ops n:ℕ
    event Init ordinary =
      thenAct n := 0
end

spec Leave over ℰ𝒱𝒯 =
  DataM0 then
  event out ordinary =
    when n < d
    thenAct n:= n+1
end

spec Enter over ℰ𝒱𝒯 =
  DataM0 then
  event in ordinary =
    when n > 0
    thenAct n := n - 1
end

spec M0ML over ℰ𝒱𝒯 =
  (Leave with out ↦ ML_out) and
  (Enter with in ↦ ML_in)
end
```

Fig. 2. This figure contains four specifications, one for the data (and the `Init` event), one for the act of leaving, one for the act of entering and the specification corresponding to the abstract Event-B machine. The separation of the abstract machine into these primitive component specifications was a design decision that we chose to make at the beginning in order to make the example more modular and to illustrate how the specification-building operators can combine these in a coherent way.

This specification contains a copy of the abstract mainland events and adds new behaviour by defining these events again here.

Lines 31–33: This is the $\mathcal{EVT}$ specification that describes the full behaviour of the Event-B machine contained in Figure 3.

It is clear that the $\mathcal{EVT}$ specification in Figure 4 corresponds to a modular version of the Event-B machine in Figure 3 but the question remains as to whether this $\mathcal{EVT}$ specification is a refinement of the abstract $\mathcal{EVT}$ specification in Figure 2. This can easily be checked using the institution theoretic notion of refinement as model class inclusion along the reduct. This broadly means that if we were to restrict the concrete specification to only contain signature elements of the abstract specification then all models of the concrete specification must also be models of the abstract. Removing the relevant signature items actually results in exactly the abstract specification given in Figure 2 and so the refinement relation holds.

1.3 The Second Refinement, M2

The next refinement step was not as gradual as the last in that quite a lot of new behaviour was added to the Event-B model as shown in Figure 5. The first big addition was that of a context that contains colours which will be used as values for the variables that are used to control the behaviour of a pair of traffic lights. Events for these lights were added to the machine and the current events were modified to account for this behaviour.

```
MACHINE m1
  refines m0
  SEES cd
  VARIABLES
    a, b, c
  INVARIANTS
     inv1: a ∈ ℕ
     inv2: b ∈ ℕ
     inv3: c ∈ ℕ
     inv4: n = a + b + c
     inv5: a = 0 ∨ c = 0
     thm1: a + b + c ∈ ℕ  theorem
     thm2: c > 0 ∨ a > 0
           ∨ (a + b < d ∧ c = 0)
           ∨ (0 < b ∧ a = 0) theorem
  VARIANT 2*a + b
  EVENTS
    Initialisation
      then
         act2: a := 0
         act3: b := 0
         act4: c := 0
    Event ML_out ≙ordinary
      refines ML_out
      when
         grd1: a + b < d
         grd2: c = 0
      then
         act1: a := a + 1
    Event IL_in ≙convergent
      when
         grd1: a > 0
      then
         act1: a := a − 1
         act2: b := b + 1
    Event IL_out ≙convergent
      when
         grd1: 0 < b
         grd2: a = 0
      then
         act1: b := b − 1
         act2: c := c + 1
    Event ML_in ≙ordinary
      refines ML_in
      when
         grd1: c > 0
      then
         act2: c := c − 1
END
```

Fig. 3. Event-B machine m1

```
spec DataM1 over 𝓔𝓥𝓣
  DataM0
  then
    ops a, b, c : ℕ
    . n = a + b + c
      a = 0 ∨ c = 0
    event Init ordinary =
      thenAct a := 0
              b := 0
              c := 0
end

spec M1IL over 𝓔𝓥𝓣 =
  DataM1 and (Enter with in ↦ IL_in, n ↦ a)
    and (Enter with in ↦ IL_out, n ↦ b)
  then
    event IL_in convergent =
      thenAct b := b + 1
    event IL_out convergent =
      when a = 0
      thenAct c := c + 1
end

spec M1ML over 𝓔𝓥𝓣 =
  DataM1 and M0ML and
    (Enter with in ↦ Ml_in, n ↦ c)
  then
   event ML_out ordinary =
     when a + b < d
          c = 0
     thenAct a := a + 1
end

spec M1 over 𝓔𝓥𝓣 =
  M1ML and M1IL
end
```

Fig. 4. This figure contains a modularised version of the $\mathcal{EVT}$-specification corresponding to the first machine at the first refinement step (m1).

Lines 1–5: This is a $\mathcal{CASL}$ specification that specifies the new data type *Color*. This corresponds to the context described in the Event-B model in Figure 5.

Lines 6–24: As in the previous specifications we have separated the data and `Initialisation` event from the rest of the specification.

Lines 25–37: This specification describes the behaviour of the refined mainland entry and exit events.

Lines 38–52: This specification describes the behaviour of the refined island entry and exit events.

Lines 53–63: This specification contains the specification of a traffic light being set to green, we separate this from the rest of the model so that we can use to account for the behaviour of two traffic lights without the need to repeat the specification twice.

Lines 64–74: This specification creates the two traffic lights that we need in accordance with the Event-B specification in Figure 5. The basic light specification that we described on lines 53–63 above is included twice here, with appropriate renamings carried out via signature morphism. This basic light was missing some information and these details have been added to each event by providing new definitions of them such that their guards and actions are merged with their previous definitions.

Lines 75–77: This is the $\mathcal{EVT}$ specification that corresponds to the full Event-B model contained in Figure 5.

1.4 The Third Refinement, M3

The third refinement step results in quite a large Event-B model as can be seen in Figures 7 and 8. We have translated this into the corresponding $\mathcal{EVT}$ specification described in Figures 9 and 10.

```
CONTEXT Color
  SETS Color
  CONSTANTS
    red, green
  AXIOMS
     axm4: Color = {green, red}
     axm3: green ≠ red
END

MACHINE m2
  refines m1
  SEES cd, Color
  VARIABLES
    a, b, c,
    ml_tl, il_tl,
    il_pass, ml_pass
  INVARIANTS
     inv1: ml_tl ∈ {red, green}
     inv2: il_tl ∈ {red, green}
     inv3: ml_tl = green ⇒ c = 0
     inv12: ml_tl = green ⇒ a + b + c < d
     inv4: il_tl = green ⇒ a = 0
     inv11: il_tl = green ⇒ b > 0
     inv6: il_pass ∈ {0, 1}
     inv7: ml_pass ∈ {0, 1}
     inv8: ml_tl = red ⇒ ml_pass = 1
     inv9: il_tl = red ⇒ il_pass = 1
     inv5: il_tl = red ∨ ml_tl = red
     thm2: 0 ≥ a ⇒ a = 0 theorem
     thm3: 0 ≥ b ⇒ b = 0 theorem
     thm4: 0 ≥ c ⇒ c = 0 theorem
     thm5: ¬(d ≤ 0) theorem
     thm6: b + 1 ≥ d ∧ ¬(b + 1 = d) ⇒ ¬(b < d)
theorem
     thm7: b ≤ 1 ∧ ¬(b = 1) ⇒ ¬(b > 0) theorem
     thm1: (ml_tl = green ∧ a + b + 1 < d)
           ∨ (ml_tl = green ∧ a + b + 1 = d)
           ∨ (il_tl = green ∧ b > 1)
           ∨ (il_tl = green ∧ b = 1)
           ∨ (ml_tl = red ∧ a + b < d ∧ c = 0 ∧ il_pass = 1)
           ∨ (il_tl = red ∧ 0 < b ∧ a = 0 ∧ ml_pass = 1)
           ∨ 0 < a ∨ 0 < c theorem
  VARIANT ml_pass + il_pass
  EVENTS
    Initialisation
      then
        act2: a := 0
        act3: b := 0
        act4: c := 0
        act1: ml_tl := red
        act5: il_tl := red
        act6: ml_pass := 1
        act7: il_pass := 1
    Event ML_out1 ≙ordinary
      refines ML_out
      when
        grd1: ml_tl = green
        grd2: a + b + 1 < d
      then
        act1: a := a + 1
        act2: ml_pass := 1
    Event ML_out2 ≙ordinary
      refines ML_out
      when
        grd1: ml_tl = green
        grd2: a + b + 1 = d
      then
        act1: a := a + 1
        act2: ml_tl := red
        act3: ml_pass := 1
    Event IL_out1 ≙ordinary
      refines IL_out
      when
        grd1: il_tl = green
        grd2: b > 1
      then
        act1: b := b − 1
        act2: c := c + 1
        act3: il_pass := 1
    Event IL_out2 ≙ordinary
      refines IL_out
      when
        grd1: il_tl = green
        grd2: b = 1
      then
        act1: b := b − 1
        act2: il_tl := red
        act3: c := c + 1
        act4: il_pass := 1
    Event ML_tl_green ≙convergent
      when
        grd1: ml_tl = red
        grd2: a + b < d
        grd3: c = 0
        grd4: il_pass = 1
      then
        act1: ml_tl := green
        act2: il_tl := red
        act3: ml_pass := 0
    Event IL_tl_green ≙convergent
      when
        grd1: il_tl = red
        grd2: 0 < b
        grd3: a = 0
        grd4: ml_pass = 1
      then
        act1: il_tl := green
        act2: ml_tl := red
        act3: il_pass := 0
    Event IL_in ≙ordinary
      refines IL_in
      when
        grd11: 0 < a
      then
        act11: a := a − 1
        act12: b := b + 1
    Event ML_in ≙ordinary
      refines ML_in
      when
        grd1: 0 < c
      then
        act1: c := c + 1
END
```

Fig. 5. Event-B machine m2 and context.

```
spec  Color  over  CASL=
  sort Color
  ops red, green : Color
  . green ≠ red
end

spec  DataM2  over  EVT =
  DataM1 and Color
  then
    ops ml_tl, il_tl:Color
        il_pass, ml_pass : {0,1}
    . ml_tl = green ⇒ c = 0
      ml_tl = green ⇒ a + b + c < d
      il_tl = green ⇒ a = 0
      il_tl = green ⇒ b > 0
      ml_tl = red ⇒ ml_pass = 1
      il_tl = red ⇒ il_pass = 1
      il_tl = red ∨ ml_tl = red
    variant ml_pass + il_pass
    event Init ordinary =
      thenAct ml_tl : = red
              il_tl : = red
              ml_pass : = 1
              il_pass : = 1
end

spec  M2ML  over  EVT =
  DataM2 and
  (M1ML hide via ML_out ↦ ML_out1) and
  (M1ML with ML_out with ML_out2)
  then
    event ML_out1 ordinary =
      when ml_tl = green
      thenAct ml_pass := 1
    event ML_out2 ordinary =
      when ml_tl = green
      thenAct ml_pass := 1
              ml_tl := red
end

spec  M2IL  over  EVT =
  DataM2 and
  (M1IL hide via IL_out ↦ IL_out1) and
  (M1IL with IL_out with IL_out2)
  then
    event IL_out1 ordinary =
      when il_tl = green
           b > 1
      thenAct il_pass := 1
    event IL_out2 ordinary =
      when il_tl = green
           b = 1
      thenAct il_pass := 1
              il_tl := red
end

spec  TLgreen  over  EVT =
  DataM2
  then
    event lgreen convergent =
      when ml_tl = red
           c = 0
           il_pass = 1
      thenAct ml_tl := green
              il_tl := red
              ml_pass := 0
end

spec  M2green  over  EVT =
  (TLgreen with lgreen ↦ ML_tl_green) and
  (TLgreen with lgreen ↦ IL_tl_green, ml_tl ↦ il_tl,
  il_pass ↦ ml_pass, il_tl ↦ ml_tl, ml_pass ↦ il_pass,
  c ↦ a)
  then
    event ML_tl_green convergent =
      when a + b < d
    event IL_tl_green convergent =
      when 0 < b
end

spec  M2  over  EVT =
  M2green and M2IL and M2ML
end
```

Fig. 6. Full description of the m2 Event-B machine using modular $\mathcal{EVT}$.

```
CONTEXT Sensor
  SETS Sensor
  CONSTANTS on, off
  AXIOMS
     axm1: Sensor = {on, off}
     axm2: ¬ on = off
END

MACHINE m3
  refines m2
  SEES  cd, Color, Sensor
  VARIABLES
    a, b, c,
    ml_tl, il_tl,
    ml_pass, il_pass,
    A, B, C,
    ML_OUT_SR, ML_IN_SR,
    IL_OUT_SR, IL_IN_SR,
    ml_out_10, ml_in_10,
    ml_in_10, il_in_10
  INVARIANTS
     inv1: IL_IN_SR = on ⇒ A > 0
     inv2: IL_OUT_SR = on ⇒ B > 0
     inv3: ML_IN_SR = on ⇒ C > 0
     inv4: ml_out_10 = TRUE ⇒ ml_tl = green

     inv5: il_out_10 = TRUE ⇒ il_tl = green
     inv6: IL_IN_SR = on ⇒ il_in_10 = FALSE

     inv7: IL_OUT_SR = on
            ⇒ il_out_10 = FALSE
     inv8: ML_IN_SR = on
            ⇒ ml_in_10 = FALSE
     inv9: ML_OUT_SR = on
            ⇒ ml_out_10 = FALSE
     inv10: il_in_10 = TRUE
            ∧ ml_out_10 = TRUE ⇒ A = a
     inv11: il_in_10 = FALSE
            ∧ ml_out_10 = TRUE ⇒ A = a + 1
     inv12: il_in_10 = TRUE
            ∧ ml_out_10 = FALSE ⇒ A = a  1
     inv13: il_in_10 = FALSE
            ∧ ml_out_10 = FALSE ⇒ A = a
     inv14: il_in_10 = TRUE
            ∧ il_out_10 = TRUE ⇒ B = b
     inv15: il_in_10 = TRUE
            ∧ il_out_10 = FALSE ⇒ B = b + 1
     inv16: il_in_10 = FALSE
            ∧ il_out_10 = TRUE ⇒ B = b  1
     inv17: il_in_10 = FALSE
            ∧ il_out_10 = FALSE ⇒ B = b
     inv18: il_out_10 = TRUE
            ∧ ml_in_10 = TRUE ⇒ C = c
     inv19: il_out_10 = TRUE
            ∧ ml_in_10 = FALSE ⇒ C = c + 1
     inv20: il_out_10 = FALSE
            ∧ ml_in_10 = TRUE ⇒ C = c  1
     inv21: il_out_10 = FALSE
            ∧ ml_in_10 = FALSE ⇒ C = c
     inv22: A = 0 ∨ C = 0
     inv23: A + B + C ≤ d
     inv24: A ∈ ℕ
     inv25: B ∈ ℕ
     inv26: C ∈ ℕ
```

```
EVENTS
  Initialisation
    then
       act2: a := 0
       act3: b := 0
       act4: c := 0
       act1: ml_tl := red
       act5: il_tl := red
       act6: ml_pass := 1
       act7: il_pass := 1
       act15: ml_out_10 := FALSE
       act16: il_out_10 := FALSE
       act17: ml_in_10 := FALSE
       act18: il_in_10 := FALSE
       act8: A := 0
       act9: B := 0
       act10: C := 0
       act11: ML_IN_SR := off
       act12: ML_OUT_SR := off
       act13: IL_OUT_SR := off
       act14: IL_IN_SR := off
    Event ML_out1 ≙ordinary
      refines ML_out1
      when
         grd1: ml_out_10 = TRUE
         grd2: a + b + 1 < d
      then
         act1: a := a + 1
         act2: ml_pass := 1
         act3: ml_out_10 := FALSE
    Event ML_out2 ≙ordinary
      refines ML_out2
      when
         grd1: ml_out_10 = TRUE
         grd2: a + b + 1 = d
      then
         act1: a := a + 1
         act2: ml_tl := red
         act3: ml_pass := 1
         act4: ml_out_10 := FALSE
    Event IL_out1 ≙ordinary
      refines IL_out1
      when
         grd1: il_out_10 = TRUE
         grd2: b > 1
      then
         act1: b := b − 1
         act2: c := c + 1
         act3: il_pass := 1
         act4: il_out_10 := FALSE
    Event IL_out2 ≙ordinary
      refines IL_out2
      when
         grd1: il_out_10 = TRUE
         grd2: b = 1
      then
         act1: b := b − 1
         act2: il_tl := red
         act3: c := c + 1
         act4: il_pass := 1
         act5: il_out_10 := FALSE
```

Fig. 7. Event-B machine description of the third refinement M3.

```
Event ML_tl_green ≙convergent
  refines ML_tl_green
  when
     grd1: ml_tl = red
     grd2: a + b < d
     grd3: c = 0
     grd4: il_pass = 1
     grd5: il_out_10 = FALSE
     grd6: ML_OUT_SR = on
  then
     act1: ml_tl := green
     act2: il_tl := red
     act3: ml_pass := 0
Event IL_tl_green ≙convergent
  refines IL_tl_green
  when
     grd1: il_tl = red
     grd2: 0 < b
     grd3: a = 0
     grd4: ml_pass = 1
     grd5: ml_out_10 = FALSE
     grd6: IL_OUT_SR = on
  then
     act1: il_tl := green
     act2: ml_tl := red
     act3: il_pass := 0
Event ML_in ≙ordinary
  refines ML_in
  when
     grd1: ml_in_10 = TRUE
     grd2: c > 0
  then
     act1: c := c − 1
     act2: ml_in_10 := FALSE
Event IL_in ≙ordinary
  refines IL_in
  when
     grd1: il_in_10 = TRUE
     grd2: 0 < a
  then
     act1: a := a − 1
     act2: b := b + 1
     act3: il_in_10 := FALSE
Event ML_OUT_ARR ≙ordinary
  when
     grd1: ML_OUT_SR = off
     grd2: ml_out_10 = FALSE
  then
     act1: ML_OUT_SR := on
Event ML_IN_ARR ≙ordinary
  when
     grd1: ML_IN_SR = off
     grd2: ml_in_10 = FALSE
     grd3: C > 0
  then
     act1: ML_IN_SR := on
Event IL_IN_ARR ≙ordinary
  when
     grd1: IL_IN_SR = off
     grd2: il_in_10 = FALSE
     grd3: A > 0
  then
     act1: IL_IN_SR := on
```

```
Event IL_OUT_ARR ≙ordinary
  when
     grd1: IL_OUT_SR = off
     grd2: il_out_10 = FALSE
     grd3: B > 0
  then
     act1: IL_OUT_SR := on
Event ML_OUT_DEP ≙ordinary
  when
     grd1: ML_OUT_SR = on
     grd2: ml_tl = green
  then
     act1: ML_OUT_SR := off
     act2: ml_out_10 := TRUE
     act3: A := A + 1
Event ML_IN_DEP ≙ordinary
  when
     grd1: ML_IN_SR = on
  then
     act1: ML_IN_SR := off
     act2: ml_in_10 := TRUE
     act3: C := C − 1
Event IL_IN_DEP ≙ordinary
  when
     grd1: IL_IN_SR = on
  then
     act1: IL_IN_SR := off
     act2: il_in_10 := TRUE
     act3: A := A − 1
     act4: B := B + 1
Event IL_OUT_DEP ≙ordinary
  when
     grd1: IL_OUT_SR = off
     grd2: il_tl = green
  then
     act1: IL_OUT_SR := off
     act2: il_out_10 := TRUE
     act3: B := B − 1
     act4: C := C + 1
end
```

Fig. 8. Event-B m3 continued.

```
spec  Sensor  over  CASL =
 sort Sensor
 ops on, off : Sensor
 . ¬ on = off
end

spec  DataM3  over  EVT =
  DataM2 and (Sensor with ρ)
  then
    ops A, B, C : ℕ
        ML_OUT_SR, ML_IN_SR : Sensor
        IL_OUT_SR, IL_IN_SR : Sensor
        ml_out_10, ml_in_10 : Bool
        il_out_10, il_in_10 : Bool
    . IL_IN_SR = on ⇒ A > 0
      IL_OUT_SR = on ⇒ B > 0
      ML_IN_SR = on ⇒ C > 0
      ml_out_10 = TRUE ⇒ ml_tl = green
      il_out_10 = TRUE ⇒ il_tl = green
      IL_IN_SR = on ⇒ il_in_10 = FALSE
      IL_OUT_SR = on ⇒ il_out_10 = FALSE
      ML_IN_SR = on ⇒ ml_in_10 = FALSE
      ML_OUT_SR = on ⇒ ml_out_10 = FALSE
      il_in_10 = TRUE ∧ ml_out_10 = TRUE
        ⇒ A = a
      il_in_10 = FALSE ∧ ml_out_10 = TRUE
        ⇒ A = a + 1
      il_in_10 = TRUE ∧ ml_out_10 = FALSE
        ⇒ A = a  1
      il_in_10 = FALSE ∧ ml_out_10 = FALSE
        ⇒ A = a
      il_in_10 = TRUE ∧ il_out_10 = TRUE
        ⇒ B=b
      il_in_10 = TRUE ∧ il_out_10 = FALSE
        ⇒ B = b + 1
      il_in_10 = FALSE ∧ il_out_10 = TRUE
        ⇒ B = b  1
      il_in_10 = FALSE ∧ il_out_10 = FALSE
        ⇒ B = b
      il_out_10 = TRUE ∧ ml_in_10 = TRUE
        ⇒ C = c
      il_out_10 = TRUE ∧ ml_in_10 = FALSE
        ⇒ C = c + 1
      il_out_10 = FALSE ∧ ml_in_10 = TRUE
        ⇒ C = c  1
      il_out_10 =FALSE ∧ ml_in_10 = FALSE
        ⇒ C = c
      A = 0 ∨ C = 0
      A + B + C ≤ d
      event Init ordinary =
        thenAct ml_out_10 := FALSE
                il_out_10 := FALSE
                ml_in_10 := FALSE
                il_in_10 := FALSE
                A := 0
                B := 0
                C := 0
                ML_IN_SR := off
                IL_IN_SR := off
                ML_OUT_SR := off
                IL_OUT_SR := off
end
```

```
spec  Toggle10  over  EVT =
  ops t: BOOL
  event toggle ordinary =
      when
        t = TRUE
      thenAct
        t = FALSE
end

spec  InOut  over  EVT =
  M2ML and M2IL
  and (Toggle10 with toggle ↦ ML_out1, t ↦ ml_out_10)
  and (Toggle10 with toggle ↦ ML_out2, t ↦ ml_out_10)
  and (Toggle10 with toggle ↦ IL_out1, t ↦ il_out_10)
  and (Toggle10 with toggle ↦ IL_out2, t ↦ il_out_10)
  and (Toggle10 with toggle ↦ ML_in, t ↦ ml_in_10)
  and (Toggle10 with toggle ↦ IL_in, t ↦ il_in_10)
end

spec  TLgreen  over  EVT =
  Sensor then
    op sensor: Sensor, b: Bool
    eventsetgreenconvergent
      when
        b = FALSE
        sensor = on
      thenAct
end

spec  M3green  over  EVT =
  M2green ikwand DataM3 and
  (TLgreen with setgreen ↦ ML_tl_green, b ↦ il_out_10,
    sensor ↦ ML_OUT_SR) and
  (TLgreen with setgreen ↦ IL_tl_green, b ↦ ml_out_10,
    sensor ↦ IL_OUT_SR)
end

spec  ARR  over  EVT =
  Sensor then
    ops sensor : Sensor, b: Bool
    event Arr ordinary =
      when
        sensor = off
        b = FALSE
      thenAct
        sensor := on
end

spec  EXTARR  over  EVT =
  ARR then
  op num : ℕ
  event Arr ordinary
    when
      num > 0
end

spec  AllARR  over  EVT =
  (ARR with Arr ↦ ML_OUT_ARR, sensor ↦ ML_out_SR,
     b ↦ ml_out_10) and
  (EXTARR with Arr ↦ ML_IN_ARR, sensor ↦ ML_in_SR,
     b ↦ ml_in_10, num ↦ C) and
  (EXTARR with Arr ↦ IL_IN_ARR, sensor ↦ IL_in_SR,
     b ↦ il_in_10, num ↦ A) and
  (EXTARR with Arr ↦ IL_OUT_ARR, sensor ↦ iL_out_SR,
     b ↦ il_out_10, num ↦ B)
end
```

Fig. 9. Full description of the m3 Event-B machine using modular $\mathcal{EVT}$.

```
spec  ILDEP  over  EVT =
  Sensor then
    ops s : sensor, b : Bool, n1,n2 : ℕ
    event ildep ordinary
      when
        s = on
      thenAct
        s := off
        b := TRUE
        n1 := n1 - 1
        n2 := n2 + 1
end

spec  AllILDEP  over  EVT =
  DataM3 and
  (ILDEP with ildep ↦ IL_IN_DEP, s ↦  IL_IN_SR,
    b ↦ il_in_10, n1 ↦ A, n2 ↦ B) and
  (ILDEP with ildep ↦ IL_OUT_DEP, s ↦  IL_OUT_SR,
    b ↦ il_out_10, n1 ↦ B, n2 ↦ C)
  then
    event IL_OUT_DEP ordinary
      when
        il_tl = green
end

spec  MLDEP  over  EVT =
  Sensor then
    ops s : Sensor, b : Bool
    event mldep ordinary
      when
        s = on
      thenAct
        s := off
        b : =TRUE
end

spec  AllMLDEP  over  EVT =
  DataM3 and
  (MLDEP with mldep ↦ ML_IN_DEP, s ↦ ML_IN_SR, b ↦ ml_in_10) and
  (MLDEP with mldep ↦ ML_OUT_DEP, s ↦ ML_OUT_SR, b ↦ ml_out_10)
  then
    event ML_IN_DEP ordinary
      thenAct
        C := C - 1
     event ML_OUT_DEP ordinary
       when
         ml_tl = green
       thenAct
         A := A + 1
end

spec  AllDEP  over  EVT =
  AllILDEP and AllMLDEP
end

spec  M3  over  EVT =
  InOut and M3green and AllARR and AllDEP
end
```

Fig. 10. The $\mathcal{EVT}$ specification corresponding to M3 continued from Figure 9