Refinement-Animation for Event-B — Towards a Method of Validation[⋆]

Stefan Hallerstede, Michael Leuschel, Daniel Plagge

Institut für Informatik, Universität Düsseldorf
Universitätsstr. 1, D-40225 Düsseldorf
{ halstefa, leuschel, plagge } @cs.uni-duesseldorf.de

Abstract. We provide a detailed description of refinement in Event-B, both as a contribution in itself and as a foundation for the approach to simultaneous animation of multiple levels of refinement that we propose. We present an algorithm for simultaneous multi-level animation of refinement, and show how it can be used to detect a variety of errors that occur frequently when using refinement. The algorithm has been implemented in ProB and we applied it to several case studies, showing that multi-level animation is tractable also on larger models.
Keywords: Refinement, Model Checking, Constraint-Solving, Tools, Industrial Applications.

1 Introduction and Motivation

The Event-B modelling method [1] has been designed to be complemented by a software tool such as Rodin [2]. The core of the Rodin tool provides automatic generation of proof obligations that can be analysed to improve understanding of a model. Often proof obligations give good indications of how to make an improvement in case of inconsistencies in a model. However, there are also many occasions where proof obligations do not point directly to a problem or where a model does not contain inconsistencies but is still "incorrect" (see, e.g., the Earley parser example discussed in [7]). In these cases animation is a useful tool to gain further insight into a model. The Rodin plugins ProB [9, 11], Brama[1] [13], and AnimB [12] provide animation facilities for Event-B.

When dealing with complex models, refinement can be used to introduce the many details gradually, achieving a reduced complexity at each refinement level. It can be difficult to analyse a refinement relationship only by means of associated proof obligations. All three animation plugins mentioned above provide some means to animate refinements. In this article we investigate their relative capabilities and how to advance refinement animation in order to turn it into a tool for refinement validation. This serves as a blueprint for the evolution of ProB in terms of animation support.

⋆ Part of this research has been EU funded FP7 project 214158: DEPLOY (Industrial deployment of advanced system engineering methods for high productivity and dependability).

[1] Brama requires an older version (0.9.2.x) of Rodin at the time of writing.

Before starting the investigation we should be clear on the objectives of animation when used for validation. What is the purpose of animating a model across multiple levels of refinement? In Event-B several concepts play a rôle in refinement. Most prominently, these are invariants, guards, actions, and witnesses. If a refinement fails, any combination of those concepts may be involved. Animation should help to locate the cause of a problem in the model pointing to specific invariants, guards, and so on, if possible. However, even if a refinement is formally correct, there can still be problems with the model. This concerns, in particular, properties that have not been formalised. Animation should make it easy to experiment with a model, visualising potential problems. We try to integrate this aspect of animation with the first one as far as possible. Otherwise consistent tool support for both would be difficult to realise.

In Section 2 we give a concise description of the fundamentals of Event-B refinement. As far as we are aware there is no single place where all the essential aspects are described and motivated in such detail. All of this is needed in order to present the basic refinement-animation algorithm in Section 3. The presentation of the algorithm is interspersed with methodical remarks on validation. In Section 4 we present some concrete examples on how to use ProB for refinement validation and some brief description of case studies to which it has been applied. Finally, Sections 5 and 6 contain a discussion of related work and a conclusion.

2 Modelling and Refinement in Event-B

Event-B can be used to model complex intricate systems. To understand the system and the model of the system we need to reason thoroughly about the model. Such reasoning is the principal purpose of Event-B. The basic concepts of Event-B are characterised by means of proof obligations; they are the core of the Event-B method. However, they are not an exclusive means of reasoning. Based on an operational interpretation of a model we can also animate it to gain deeper understanding. In this section we parallel the presentation of Event-B proof obligations, in particular, refinement, with ideas of animation. This demonstrates well how animation complements proof. Because there is no single software tool for animation that provides all that is needed, we use the three tools ProB, Brama, and AnimB at the same time.

2.1 Contexts

Event-B models are described in terms of the two basic constructs: *contexts* and *machines*. Contexts specify static parts of a model, that is, carrier sets and constants that are constrained by axioms. Usually, these are quite simple formulas. Contexts are intended to be used to parametrise machines. We mention contexts here because of the rôle they play in animation. For any particular animation specific values for all constants have to be found. ProB does this automatically using constraint-solving techniques to find proper values that satisfy all axioms. The constraint-solving also determines whether the axioms contain a contradiction.

2.2 Machines

Machines provide behavioural properties of Event-B models. Machines may contain *variables*, *invariants*, *events*, and *variants*. Variables v define the state of a machine. They are constrained by invariants $I(v)$. Possible state changes are described by means of events. Each event is composed of a *guard* $G(t, v)$ and an *action* $S(t, v)$, where t are *parameters* of the event. The guard states the necessary condition under which an event may occur, and the action describes how the state variables evolve when the event occurs. We denote an event $E(v)$ by one of the following forms:

```
any  $t$  when          or          when          or          begin
  $G(t,v)$                             $G(v)$                          $S(v)$
then                                then                          end
  $S(t,v)$                             $S(v)$
end                                 end
```

The second form is used if event $E(v)$ does not have parameters, and the third form if in addition the guard equals *true*. A dedicated event of the third form is used for *INITIALISATION*. In the formal exposition below, we assume without loss of generality that the most general first form is used.

The action of an event is composed of several *assignments* of the form: $x := E(t, v)$ or $x :\in E(t, v)$ or $x :\mid Q(t, v, x')$, where x are some variables, $E(t, v)$ expressions, and $Q(t, v, x')$ a predicate. The second form assigns x to an element of a set, and the third form assigns to x a value satisfying a predicate. Without loss of generality, the first two can be formally defined in terms of the third form: $x := E(t, v) \mathrel{\widehat{=}} x :\mid x' = E(t, v)$ and $x :\in E(t, v) \mathrel{\widehat{=}} x :\mid x' \in E(t, v)$. The effect of an assignment is described by a before-after predicate:

$$\text{before-after predicate of "}x :\mid Q(t, v, x')\text{"} \quad \widehat{=} \quad Q(t, v, x')$$

A before-after predicate describes the relationship between the state just before an assignment has occurred, x, and the state just after the assignment has occurred, x'. All assignments of an action $S(t, v)$ occur simultaneously which is expressed by conjoining their before-after predicates, yielding a predicate $A(t, v, x')$. Variables y that do not appear on the left-hand side of an assignment of an action are not changed by the action. Formally, this is achieved by conjoining $A(t, v, x')$ with $y' = y$, yielding the predicate:

$$\boldsymbol{S}(t, v, v') \quad \widehat{=} \quad A(t, v, x') \wedge y' = y \quad .$$

Running Example. We use the coffee dispenser model in Fig. 1 for illustration of refinement-animation. In the abstract machine *CoffeeM* the dispenser can fill a mug half or fully; the state of the mug is represented by the variable *alvl* (abstract level). As a special service the dispenser can also drink the coffee. In the first refined machine *CoffeeR1* a feature is introduced for inserting an arbitrary

```
machine CoffeeM sees CofCtxt
variables alvl
invariants @inv1 alvl ∈ FILL
variant ({full ↦ 2, half ↦ 1, empty ↦ 0})(alvl)
events
  event INITIALISATION
    begin
      @mf alvl := empty
    end
  event fill_mug
    any x when
      @g0 alvl = empty
      @g1 x ≠ alvl
    then
      @a1 alvl := x
    end
  convergent event drink
    when @g1 alvl ≠ empty then
      @a1 alvl :∈ {empty, half} \ {alvl}
    end
end
```

```
context CofCtxt
constants full empty half level
sets FILL
axioms
  @Ffhe partition(FILL, {full}, {half}, {empty})
  @lvl level = (0 .. 2 × {empty}) ∪ (3 .. 7 × {half}) ∪ (8 .. 11 × {full})
end
```

```
machine CoffeeR1 refines CoffeeM sees CofCtxt
variables alvl coins
invariants @ci coins ∈ ℕ
events
  event INITIALISATION extends INITIALISATION
    begin
      @ai coins := 0
    end
  event fill_mug extends fill_mug
    when @gc coins > 0 then
      @delc coins := coins − 1
    end
  convergent event drink extends drink
  end
  anticipated event insert_coin
    begin
      @insc coins := coins + 1
    end
end
```

```
machine CoffeeR2 refines CoffeeR1 sees CofCtxt
variables clvl coins maxc
invariants
  @imc maxc ∈ ℕ1
  @ifl clvl ∈ 0 .. 11
  @lvl alvl = level(clvl)
variant maxc − coins
events
  event INITIALISATION
    begin
      @mc maxc := 4
      @cci coins := 0
      @fli clvl := 0
    end
  event fill_mug refines fill_mug
    when
      @gc2 coins > 0
      @ml clvl ∈ level^{-1}[{empty}]
    with
      @x x = level(clvl')
    then
      @delc2 coins := coins − 1
      @ffl clvl :∈ level^{-1}[{full}]
    end
  convergent event drink refines drink
    when
      @dgfl clvl ∉ level^{-1}[{empty}]
    with
      @alvl' alvl' = level(clvl')
    then
      @dfl clvl :∈ level^{-1}[{empty, half} \ {level(clvl)}]
    end
  convergent event insert_coin extends insert_coin
    when
      @gmc coins < maxc
    end
end
```

Fig. 1: Coffee dispenser model (using syntax of the Event-B text editor "Camille")

number of coins into the dispenser. A coin is consumed each time a mug is filled. In the second refined machine *CoffeeR2*, the number of coins maximally accepted is limited and the amount of coffee contained in a mug is represented numerically by a the variable *clvl* (concrete level).

Animation in ProB. The before-after predicate can be used to compute the state space of a machine, a graph where each node represents a state of the machine and each arc the execution of an event. Fig. 2 contains the state space of the *CoffeeM* machine from Fig. 1, as computed by the ProB tool. The triangle represent a special root node, where the variables and constants of a machine have not yet been set. An animator lets the user navigate

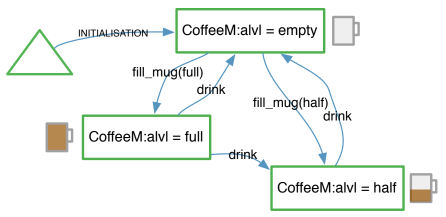


Fig. 2: State space for *CoffeeM* (with additional mugshots)

the state space by choosing the events to be fired. A model checker will systematically explore the state space, looking for various errors in the machine.

2.3 Machine Consistency

Invariants are supposed to hold initially and whenever variable values are changed by an event. Obviously, this does not hold a priori and, thus, needs to be proved. The corresponding proof obligation for every event is called *invariant preservation*, formally, $I(v) \wedge G(t,v) \wedge \boldsymbol{S}(t,v,v') \Rightarrow I(v')$. There is a special form of this proof obligation, without invariant and guard in the hypothesis, for the *INITIALISATION*. By proving *action feasibility* for an event, $I(v) \wedge G(t,v) \Rightarrow (\exists v' \cdot \boldsymbol{S}(t,v,v'))$, as well, we achieve that $\boldsymbol{S}(t,v,v')$ provides an after state whenever $G(t,v)$ holds. This means that the guard indeed represents the enabling condition of the event.

2.4 Machine Refinement

A machine N can refine at most one other machine M. We call M the *abstract* machine and N a *concrete* machine. The state of the abstract machine is related to the state of the concrete machine by a *gluing invariant* $J(v,w)$ associated with the concrete machine N, where v are the variables of the abstract machine and w the variables of the concrete machine.

Each event $E(v)$ of the abstract machine is *refined* by one or more concrete events $F(w)$. Let abstract event $E(v)$ and concrete event $F(w)$ be:

$$E(v) \quad \widehat{=} \quad \mathsf{any}\ t\ \mathsf{when}\ G(t,v)\ \mathsf{then}\ S(t,v)\ \mathsf{end}$$
$$F(w) \quad \widehat{=} \quad \mathsf{any}\ u\ \mathsf{when}\ H(u,w)\ \mathsf{with}\ W(t,v',u,v,w,w')\ \mathsf{then}\ T(u,w)\ \mathsf{end}$$

Informally, concrete event $F(w)$ refines abstract event $E(v)$ if, whenever the gluing invariant $J(v,w)$ is true: (i) the guard of $F(w)$ is stronger than the guard of $E(v)$, and (ii) for every possible execution of $F(w)$ there is a corresponding execution of $E(v)$ which simulates $F(w)$ such that the gluing invariant remains true after execution of both events. In Fig. 1 some events carry the attribute "extended". This means that all parameters, guards, and actions are copied literally from the abstract event.[2] Note that the event $F(w)$ contains one more component $W(t,v',u,v,w,w')$ following the keyword with, called the *witnesses*. We return to its rôle in Section 2.6 below.

Refinement Animation. To check whether the guard of a concrete event is stronger, we also need to animate the corresponding abstract machine. Fig. 3 shows a graphical visualisation (created with Brama) of an animation of the coffee dispenser model described earlier. Green boxes signal en-

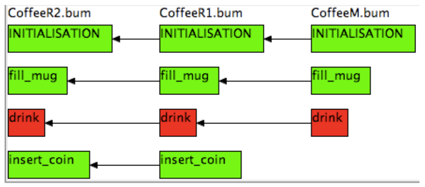
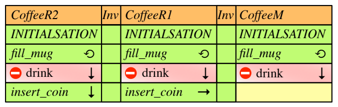


Fig. 3: Coffee dispenser refinement-animation in Brama

[2] In Event-B it is also possible for an event to refine more than one abstract event, merging these events into one concrete event [3]. Such events cannot be extended.

abled events, red boxes disabled events. As can be seen, all the refinement levels are animated concurrently. Brama's representation also shows at a glance that whenever a refined event is enabled, then all of its ancestor events are also enabled.

The view underlying Fig. 3 is operational. It focuses solely on event execution. If we wanted to use it for analysing a formal model, we would need to add information. In particular, information about gluing invariants would be useful (Fig. 4). Note, that this is not a purely cosmetic change: the animator must supply all necessary information.



CoffeeR2	*Inv*	*CoffeeR1*	*Inv*	*CoffeeM*
INITIALSATION		*INITIALSATION*		*INITIALSATION*
fill_mug ⟲		*fill_mug* ⟲		*fill_mug* ⟲
⛔ drink ↓		⛔ drink ↓		⛔ drink ↓
insert_coin ↓		*insert_coin* →		

Fig. 4: Improved coffee dispenser refinement-animation

2.5 Common Variables and Common Parameters

As far as animation and model checking are concerned, refinement introduces a new challenge: we no longer have just a single machine that needs to be animated as in Fig. 2, but a series of machines, each with its own state.[3]

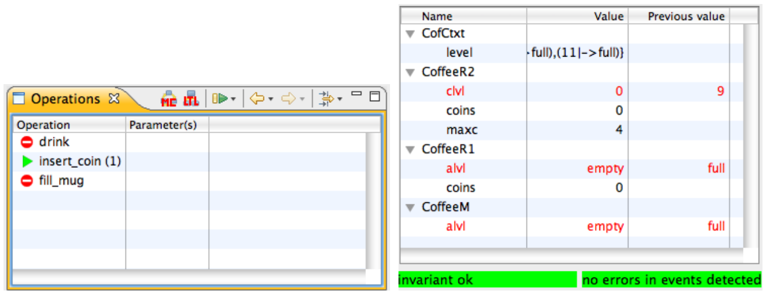
In order to check the gluing invariant, we need to access variables from various machines. This raises a new issue. In Section 2.4 we have simply assumed that all variables v are refined by new variables w, and all parameters t are refined by new parameters u. The variables v and parameters t "disappear" in the refinement. In practice, variables and parameters can be repeated in a refinement. Abstract machines and concrete machines can have variables in common, and abstract events and concrete events parameters. By convention, when repeating variables and parameters abstract and concrete counterparts are assumed to be equal.[4]

Animation. For refinement animation of machines this means that variables must be renamed in each machine and gluing invariants generated. If the animation would operate on variables shared between different machines, it would not be possible to visualise machines with deviating behaviour. This also affects the witnesses described in the following section.

Example. In the example from Fig. 1, the machine *CoffeeM* and *CoffeeR1* have the variable *alvl* in common. Fig. 5 shows ProB animating the Coffee example. As can be seen in the newly developed hierarchical "State View", the variable *alvl* occurs twice, once in *CoffeeM* and once in *CoffeeR1*. We can also see that the variable *alvl* disappears when going to *CoffeeR2*. In Fig. 6 we show how the AnimB animator displays a state of multiple refinement-levels; each refinement level is given its own tab.

[3] Earlier versions of ProB avoided this problem by animating each refinement level separately, at the cost of not being able to check the gluing invariant and of less user feedback.

[4] Once a variable has disappeared in the course of several refinements it cannot reappear. The reason for this is that the equality cannot be established by means of a machine that does not contain the variable. Furthermore, invariants are accumulated in Event-B. So it is not possible to reintroduce a variable with a different meaning.



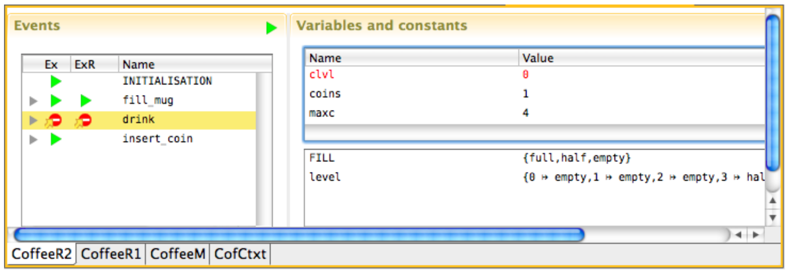
Operations

Operation	Parameter(s)
⛔ drink	
▶ insert_coin (1)	
⛔ fill_mug	

Name	Value	Previous value
▼ CofCtxt		
level	·full),(11|->full)}	
▼ CoffeeR2		
clvl	0	9
coins	0	
maxc	4	
▼ CoffeeR1		
alvl	empty	full
coins	0	
▼ CoffeeM		
alvl	empty	full

invariant ok | no errors in events detected

Fig. 5: ProB Operations and State View after the trace *insert_coin*, *fill_mug*, *drink*



Events

Ex	ExR	Name
▶		INITIALISATION
▶	▶	fill_mug
⛔	⛔	drink
▶		insert_coin

Variables and constants

Name	Value
clvl	0
coins	1
maxc	4

FILL	{full,half,empty}
level	{0 ↦ empty,1 ↦ empty,2 ↦ empty,3 ↦ hal

CoffeeR2 | CoffeeR1 | CoffeeM | CofCtxt

Fig. 6: AnimB View after *insert_coin*,*insert_coin*, *fill_mug*, *drink*

2.6 Refined Events and Witnesses

The predicate $W(t, v', u, v, w, w')$ denotes *witnesses*. Somewhat simplified, they link the abstract parameters t and the abstract variables v' to concrete parameters u and variables and w' (see also Fig. 7). Witnesses describe for each event separately how the refinement is achieved. Let $K(v, w) \;\widehat{=}\; I(v) \wedge J(v, w)$.

Aside. As described in [4], in order to verify that $F(w)$ refines $E(v)$ we have to prove $K(v, w) \wedge H(u, w) \wedge \mathbf{T}(u, w, w') \Rightarrow \exists t, v' \cdot G(t, v) \wedge \mathbf{S}(t, v, v') \wedge J(v', w')$. In a proof of this statement we prefer to instantiate the quantified parameters and variables t and v' by expressions that can in some way inferred from the premises. This idea is generalised to the witnesses used in Event-B. Witnesses are predicates that provide values to satisfy the conclusion of the statement.

The proof obligations for concrete machines are called *guard strengthening*: $K(v, w) \wedge H(u, w) \wedge \mathbf{T}(u, w, w') \wedge W(t, v', u, v, w, w') \Rightarrow G(t, v)$, *action simulation*: $K(v, w) \wedge H(u, w) \wedge \mathbf{T}(u, w, w') \wedge W(t, v', u, v, w, w') \Rightarrow \mathbf{S}(t, v, v')$, and *invariant preservation*: $K(v, w) \wedge H(u, w) \wedge \mathbf{T}(u, w, w') \wedge W(t, v', u, v, w, w') \Rightarrow J(v', w')$. We have to prove *witness feasibility* in order to be able to add the witness predicate $W(t, v', u, v, w, w')$ to the premises in the proof obligations above: $K(v, w) \wedge H(u, w) \wedge \mathbf{T}(u, w, w') \Rightarrow (\exists t, v' \cdot W(t, v', u, v, w, w'))$.

In general, witnesses would be required for all parameters p of an event but when a parameter is repeated in a refined event, by convention, it is assumed to be equal to the corresponding abstract parameter. If a parameter is not repeated an explicit witness is required. (The Rodin tool creates the *default witness* "*true*" if none is specified in the latter case. This witness does not constrain the relationship between abstract and concrete parameters and variables. Hence, usually default witnesses are not sufficient to establish the refinement relationship.) For variables the rule when witnesses are needed is more complicated: whenever a variable x that disappears occurs in a non-deterministic assignment in the abstract event, in the refined event a witness for the post-state variable v' is required.

Animation. For animation we have to take care that as a consequence of variable and parameter renaming (resulting from repeated variables), some witnesses may have to be generated. Combined with the generated gluing invariant (as described in Section 2.5) they provide an opportunity to locate refinement mismatches and provide meaningful feedback to the user.

Example. Event *fill_mug* in *CoffeeR2* contains the witness $x = level(clvl')$ for the abstract parameter x of *fill_mug* in *CoffeeR1*. This means intuitively, that every execution of *fill_mug* in *CoffeeR2* corresponds to an execution of *fill_mug* in *CoffeeR1* with parameter x set to $level(clvl')$. Event *fill_mug* in *CoffeeR1* must be enabled for $x = level(clvl')$ and the gluing invariant $alvl = level(clvl)$ must hold after executing the abstract and concrete event. Similarly, *drink* in *CoffeeR2* contains a witness $alvl' = level(clvl')$ for the abstract variable $alvl$. (Note, that it is just invariant @lvl in Fig. 1 with all variables primed.)

Animation in ProB***.*** Witnesses are the key concept that makes refinement animation possible. Indeed, refinement animation and refinement checking in classical B require for every concrete state to keep track of the *set* of all abstract states for which the gluing invariant holds. Only if this set becomes empty, have we found an error in the refinement. The size of state space necessary for simulation grows exponentially. In Event-B, by contrast, the witnesses pinpoint the states which have to satisfy the refinement relationship (see Fig. 7).

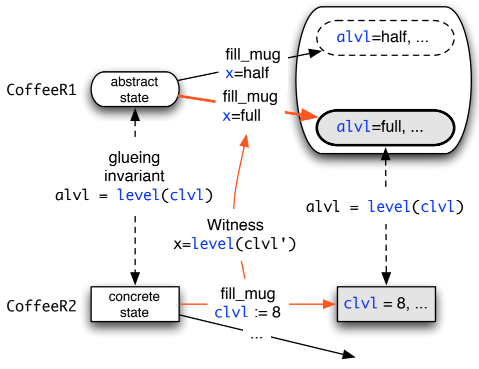


Fig. 7: Witnesses and Multi-Level Animation

2.7 New Events and Convergence

In the course of refinement, often *new events* $F(w)$ are introduced into a model. New events must be proved to refine the implicit abstract event *skip* that does nothing. Moreover, it may be proved that new events do not collectively diverge by proving that a specified *variant* $V(w)$ is bounded from below: $K(v,w) \land H(u,w) \Rightarrow V(w) \geq 0$ and is decreased by each new event $K(v,w) \land H(u,w) \land \boldsymbol{T}(u,w,w') \Rightarrow V(w') < V(w)$ where we assume that the variant is an integer expression. (Instead of an integer expression also a finite set expression can be used.) We call events that satisfy these two proof obligations *convergent*.

Anticipated events can be used to prove convergence on a lexicographic order or just to delay convergence proofs. Anticipated events can be refined by anticipated or convergent events, but must ultimately be refined by a convergent event. For an anticipated event the second proof obligation is replaced by $K(v, w) \wedge H(u, w) \wedge \boldsymbol{T}(u, w, w') \Rightarrow V(w') \leq V(w)$.
Example. Event insert_coin in Fig 1 is anticipated in CoffeeR1 and is then proven convergent in CoffeeR2 by introducing an upper bound on the number of inserted coins.

2.8 Enabledness of Refined and New Events

We may prove that whenever the abstract machine may continue by means of event $E(v)$ with guard $G(t, v)$ then the concrete machine may continue by means of concrete event $F(w)$ or some other events $F_1(w), \ldots, F_k(w)$, $K(v, w) \wedge G(t, v) \Rightarrow (\exists u \cdot H(u, w)) \vee (\exists u_1 \cdot H_1(u_1, w)) \vee \ldots \vee (\exists u_k \cdot H_k(u_k, w))$. The Rodin tool does not support enabledness proof obligations at the moment. But ProB supports analysis of liveness properties and animation can show a deadlock (where all events except for the initialisation are disabled).

3 Description of the Multi-Level Animation Algorithm

In this section we describe the validation and animation algorithm in detail. We point out in the presentation of the algorithm how it indicates problems with particular proof obligations. We also show how feedback to the user needs to be considered. Producing informative output from an animation with good performance is a challenge. For this reason the algorithm makes heavy use of ProB's existing functionality. In particular, ProB provides methods to find values for variables that satisfy predicates occurring in Event-B models.

Below we limit discussion to animation; but the algorithm is identical for model checking: the model checker uses the same technique to determine the state space.

3.1 Preprocessing

The algorithm is applied to a particular refinement machine M_i of a model. In a pre-processing step, all ancestor machines $M_0, \ldots, M_{i-1}$ of M_i are loaded and all contexts seen by $M_0, \ldots, M_i$ are merged by collecting the declared constants and joining the axioms. All variables and constants are tagged according to the model or context where they are defined. The invariant is obtained by conjoining all invariants of $M_0, \ldots, M_i$.

We transform each event of M_i to an internal representation. The representation is outlined on the right hand side of Fig. 8. Usually the list of abstract events contains just one entry. If an event refines *skip* or belongs to the most abstract machine M_0, the list of abstract events is empty. If the event refines several events, it will contain all of those events.

3.2 The Animation Algorithm

The animator executes events depending on the current state of a model. It maintains a state consisting of all constants of the seen contexts as well as all variables of the machines $M_0, \ldots, M_i$.

In a first step the animator tries to find values for the constants that satisfy all axioms. Subsequently, the animator executes in each step an event of the most concrete machine M_i; then it executes the corresponding abstract events from the concrete event to the most abstract event. When all of this has been done, the animator is ready for the next step.

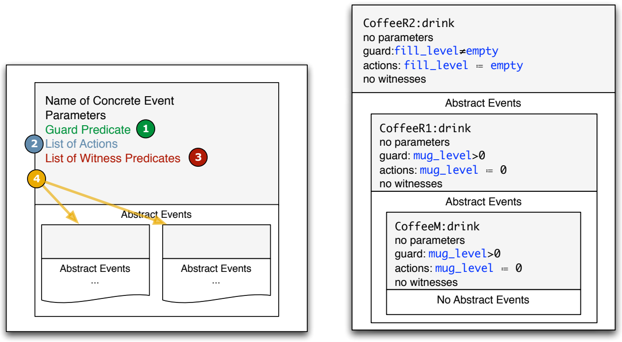


Fig. 8: Illustration of the algorithm and one particular event structure

The algorithm to animate a particular event works as follows (item numbers correspond to those of Fig. 8, left hand side):[5]

1. Search for possible values for the parameters by evaluating its guard. If no values are found, the event is disabled.
2. Execute each action by evaluating the respective before-after predicate. If no solution is found, report an error. The possible reasons for a failing action are:
   a. The predicate P of an action $v \ :\mid \ P$ is not satisfiable or the set S of an action $v \ :\in \ S$ is empty. Both cases show violations of the event feasibility proof obligation.
   b. The new value v' of a variable v was previously determined by a witness of a refined event (see step 3), but the abstract action cannot assign the same value to v'. This indicates a violation of the action simulation proof obligation.
3. For each witness evaluate its predicate and try to find values for the witnessed variable. If no value is found for a witness, report an error, because a witness should have at least one solution (by the witness feasibility proof obligation).
4. a. If the list of abstract events is empty, a complete solution has been found for this event that leads to a new state. It consists of the values newly assigned by the actions plus the variables unchanged by the actions.
   b. If there are one or more abstract events, choose one nondeterministically and evaluate its guard like in step 1. If it evaluates to true, continue recursively with step 2, otherwise try the next event.

[5] With respect to animation *INITIALISATION* is not treated differently from any other events (see left of Fig. 8) except that it is enforced to occur once upon start of an animation.

If no guard evaluates to true, report an error, because the guard of the refinement is weaker than that of the abstract event (violation of the guard strengthening proof obligation).

All four steps can be nondeterministic and we generate all solutions (limited to a maximum number) with backtracking.

Animation of convergent and anticipated events. If we have successfully found a possible event leading from one state to another, we can easily check if the convergence criteria are satisfied. The principle is quite simple: for each convergent event of an animated model, we check if the variant V is decreased and non-negative by the predicates $V > V'$ and $V \geq 0$ resp. $V \supset V'$ when the variant is a set. If the event refines another convergent event, we omit the test because in the lexicographic order constructed by refinement, events that have been shown to decrease a variant in an abstraction of some concrete machine may increase the variant of the concrete machine. Similarly, we can check anticipated events (with $V \geq V'$ and $V \geq 0$ resp. $V \supseteq V'$), but we cannot omit the test if an event is a refinement of an anticipated event.

Animating only a part of the refinement chain. Above we presumed that the user wants to animate a refinement M_i and all its ancestors $M_0, \ldots, M_{i-1}$. But we also permit the user to limit the animation to the refinements between M_i and an "upper" refinement M_k with $0 \leq k \leq i$ instead of M_0. Then variables of not animated models and predicates that contain references to those variables will be removed.

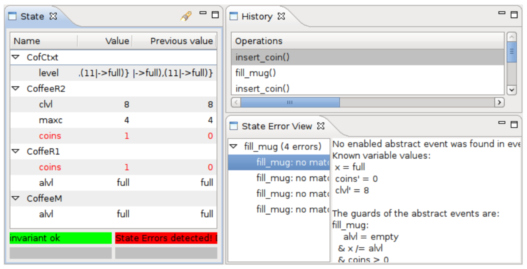
4 Refinement-Validation with ProB

Refinement animation can be used to validate models. We present some specific problems that can be analysed by animation and discuss a selection of case studies to which it has been applied to.

4.1 Detection of specific problems

Below we show on various modified versions of the coffee model (Fig. 1), how the new multi-level animation algorithm allows ProB to detect a variety of refinement errors. Note that in contrast to AnimB and Brama, ProB can be driven by a model checker so as to systematically detect refinement errors.

Guard Weakening. If we remove the guard @ml from the event *fill_mug* in *CoffeeR2*, we violate the guard strengthening proof obligation. As can be seen in Fig. 9, ProB's model checker using our new algorithm detects this problem straightaway (case **??** of our algorithm), leading us to



Name	Value	Previous value
CofCtxt		
level	,(11|->full)}	|->full),(11|->full)}
CoffeeR2		
clvl	8	8
maxc	4	4
coins	1	0
CoffeR1		
coins	1	0
alvl	full	full
CoffeeM		
alvl	full	full

invariant ok
State Errors detected!

```
Operations
insert_coin()
fill_mug()
insert_coin()
```

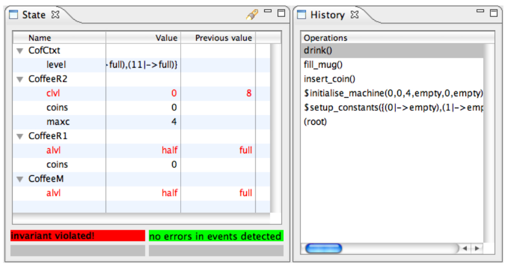
```
fill_mug (4 errors)
    fill_mug: no matc
    fill_mug: no matc
    fill_mug: no matc
    fill_mug: no matc
```

```
No enabled abstract event was found in eve
Known variable values:
 x = full
 coins' = 0
 clvl' = 8

The guards of the abstract events are:
fill_mug:
   alvl = empty
 & x /= alvl
 & coins > 0
```

Fig. 9: Violation of Guard Strengthening (ProB)

a state where *fill_mug* is enabled in *CoffeeR2* but not in the abstract machines.
Witness disables abstract guard. A similar error message appears if we keep the guards as they are, but inject an error in the witness. E.g, when using $x = empty$ as witness for *fill_mug*, ProB detects that there is a solution for the witness, but that the witness does not enable the abstract event.
Witness not feasible. Next, let us use the witness $x = level(clvl') \wedge x = empty$ for event *fill_mug*. Here case (3) of our algorithm detects an error for *fill_mug* (after executing *insert_coin*), and ProB displays the error message: "No solution found for witness of the abstract parameter x in event CoffeeR2:fill_mug". The animator AnimB does not detect this error (but it did detect the previous two errors).
Witness violates invariant. Finally, we try to specify the witness $alvl' \in \{empty, half\} - \{alvl\}$ for the event drink, which does not guarantee that the abstract event will satisfy the gluing invariant. As can be seen in Fig. 10, ProB finds an invariant violation error ($alvl = level(clvl)$ is false) directly after the drink event.



State

Name	Value	Previous value
CofCtxt		
level	·full),(11|->full)}	
CoffeeR2		
clvl	0	8
coins	0	
maxc	4	
CoffeeR1		
alvl	half	full
coins	0	
CoffeeM		
alvl	half	full

invariant violated!

no errors in events detected

History

Operations
drink()
fill_mug()
insert_coin()
$initialise_machine(0,0,4,empty,0,empty)
$setup_constants({(0|->empty),(1|->emp

(root)

Fig. 10: Violation of Gluing Invariant (ProB)

Note that, AnimB detects an error in the model, but only later when trying to execute the fill_mug event after the erroneous drink event.

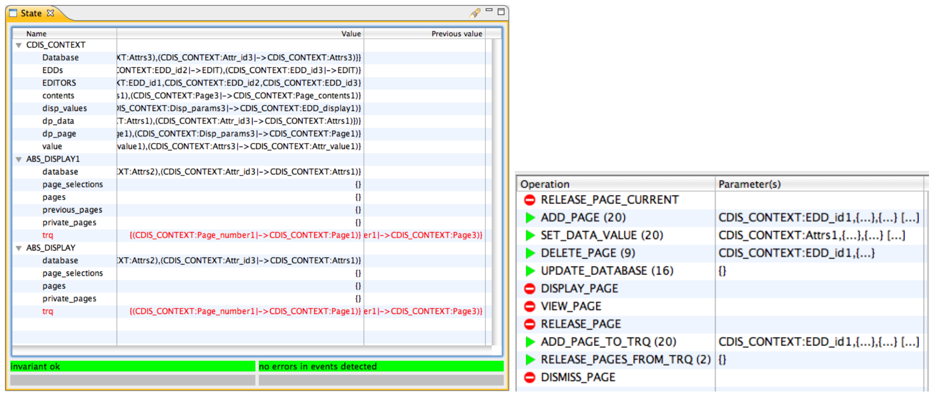
In practice, validation by animation complements the proof-based methodology of core Event-B. Corresponding methodological benefits of using animation of Event-B models are discussed in more detail in [8].

4.2 Application to case studies

We have successfully applied the new multi-level animation of ProB on a two-level model of SAP service choreographies [14]. We have also tested the tool on the CDIS air traffic control case study carried out in the EU project Rodin. Figure 11 contains a screenshot of the first two levels; we have successfully animated all 7 levels of the full model concurrently.

Another case study was a complete development of the quicksort algorithm in Event-B, consisting of ten machines and two contexts. We have successfully animated and model checked all the ten levels concurrently. Animation showed how the algorithm "works" at different abstraction levels. This is valuable for explaining an otherwise static model of an algorithm.

We have also successfully animated concurrently 14 levels of an elevator model solution by ETH Zürich. This has uncovered a potential problem in the model, namely that starting at a certain refinement level, the lift is no longer able to move (but the doors can be opened and closed and the buttons can be pressed; so there is no deadlock in the conventional sense).



Name	Value	Previous value
CDIS_CONTEXT		
Database	XT:Attrs3),(CDIS_CONTEXT:Attr_id3|->CDIS_CONTEXT:Attrs3))}	
EDDs	ONTEXT:EDD_id2|->EDIT),(CDIS_CONTEXT:EDD_id3|->EDIT)}	
EDITORS	XT:EDD_id1,CDIS_CONTEXT:EDD_id2,CDIS_CONTEXT:EDD_id3}	
contents	s1),(CDIS_CONTEXT:Page3|->CDIS_CONTEXT:Page_contents1)}	
disp_values	IS_CONTEXT:Disp_params3|->CDIS_CONTEXT:EDD_display1)}	
dp_data	T:Attrs1),(CDIS_CONTEXT:Attr_id3|->CDIS_CONTEXT:Attrs1)})}	
dp_page	ge1),(CDIS_CONTEXT:Disp_params3|->CDIS_CONTEXT:Page1)}	
value	value1),(CDIS_CONTEXT:Attrs3|->CDIS_CONTEXT:Attr_value1)}	
ABS_DISPLAY1		
database	XT:Attrs2),(CDIS_CONTEXT:Attr_id3|->CDIS_CONTEXT:Attrs1)}	
page_selections	{}	
pages	{}	
previous_pages	{}	
private_pages	{}	
trq	{(CDIS_CONTEXT:Page_number1|->CDIS_CONTEXT:Page1)}	er1|->CDIS_CONTEXT:Page3)}
ABS_DISPLAY		
database	XT:Attrs2),(CDIS_CONTEXT:Attr_id3|->CDIS_CONTEXT:Attrs1)}	
page_selections	{}	
pages	{}	
private_pages	{}	
trq	{(CDIS_CONTEXT:Page_number1|->CDIS_CONTEXT:Page1)}	er1|->CDIS_CONTEXT:Page3)}

invariant ok

no errors in events detected

Operation	Parameter(s)
⛔ RELEASE_PAGE_CURRENT	
▶ ADD_PAGE (20)	CDIS_CONTEXT:EDD_id1,{...},{...} [...]
▶ SET_DATA_VALUE (20)	CDIS_CONTEXT:Attrs1,{...},{...} [...]
▶ DELETE_PAGE (9)	CDIS_CONTEXT:EDD_id1,{...}
▶ UPDATE_DATABASE (16)	{}
⛔ DISPLAY_PAGE	
⛔ VIEW_PAGE	
⛔ RELEASE_PAGE	
▶ ADD_PAGE_TO_TRQ (20)	CDIS_CONTEXT:EDD_id1,{...},{...} [...]
▶ RELEASE_PAGES_FROM_TRQ (2)	{}
⛔ DISMISS_PAGE	

Fig. 11. Animating the Rodin CDIS case study

5 Related Work

As already indicated above, the tools Brama and AnimB are also capable of performing multi-level animation of Event-B models, and have partially inspired this work. Unfortunately there is little scientific or technical documentation available for both of these tools. A few notable differences are

- Both Brama and AnimB require to specify explicitly values for constants; i.e., we had to "calculate" the cartesian products for the level constant in Fig. 1 by hand.
- ProB can be driven by a model checker to systematically search for errors, and to validate LTL formulas.
- ProB uses a constraint solving approach to find solutions for predicates, while AnimB and Brama seem to rely on pure enumeration. As such, ProB can evaluate much more complicated guards and predicates than AnimB or Brama.

Another animator for Event-B is [5]; but it does not yet seem to support refinement animation. The same is true for the animator in [6] for classical B. Another related work is the refinement checking algorithm in [10]. This algorithm does not have access to Event-B's witnesses and hence has to keep track of sets of states in the abstract model (and does not check the gluing invariant as the traces of the abstract and refined model are computed separately).

6 Conclusion

We have presented a description of refinement in Event-B and have shown how a suitable animation and validation algorithm can be developed. The key ingredient that makes the algorithm tractable are the witnesses of Event-B. We have implemented the algorithm within ProB, and have shown how a variety of refinement errors can now be detected effectively. We have applied the technique to various case studies, and have animated up to 14 levels simultaneously.

In future work, we plan combining the graphical representation of Brama of Fig. 3 with the validation features of ProB. As we have sketched in Fig. 4, we also would like to be able to see when an event is disabled in a concrete machine but enabled in an abstract machine (Brama does not compute this information), and also to visualize the gluing invariant of each refinement level individually. We would also like to visualise the errors found by ProB inside the Rodin models, e.g., so that the offending proof obligations can be marked as "not provable."

References

1. J.-R. Abrial. *Modeling in Event-B: System and Software Engineering.* Cambridge University Press, 2009. To appear.
2. J.-R. Abrial, M. Butler, S. Hallerstede, and L. Voisin. An open extensible tool environment for Event-B. In *ICFEM06*, LNCS 4260, pages 588–605, 2006.
3. J.-R. Abrial, D. Cansell, and D. Méry. Refinement and Reachability in EventB. In H. Treharne, S. King, M. Henson, and S. Schneider, editors, *ZB 2005*, volume 3455 of *LNCS*, pages 222–241, 2005.
4. J.-R. Abrial and S. Hallerstede. Refinement, decomposition, and instantiation of discrete models: Application to event-B. *Fundam. Inform*, 77(1-2):1–28, 2007.
5. I. Aït-Sadoune and Y. A. Ameur. Animating event b models by formal data models. In T. Margaria and B. Steffen, editors, *ISoLA*, volume 17 of *Communications in Computer and Information Science*, pages 37–55. Springer, 2008.
6. F. Ambert, F. Bouquet, S. Chemin, S. Guenaud, B. Legeard, F. Peureux, M. Utting, and N. Vacelet. BZ-testing-tools: A tool-set for test generation from Z and B using constraint logic programming. In *Proceedings of FATES'02*, pages 105–120, August 2002. Technical Report, INRIA.
7. J. Bendisposto, M. Leuschel, O. Ligot, and M. Samia. La validation de modèles Event-B avec le plug-in ProB pour RODIN. *Technique et Science Informatiques*, 27(8):1065–1084, 2008.
8. S. Hallerstede and M. Leuschel. How to explain mistakes. In J. Gibbons and J. N. Oliveira, editors, *TFM 2009*, LNCS 5846, pages 105–124. Springer, 2009.
9. M. Leuschel and M. Butler. ProB: A model checker for B. In K. Araki, S. Gnesi, and D. Mandrioli, editors, *FME 2003*, LNCS 2805, pages 855–874, 2003.
10. M. Leuschel and M. Butler. Automatic refinement checking for B. In K.-K. Lau and R. Banach, editors, *ICFEM'05*, LNCS 3785, pages 345–359, 2005.
11. M. Leuschel and M. J. Butler. ProB: an automated analysis toolset for the B method. *STTT*, 10(2):185–203, 2008.
12. C. Métayer. http://www.animb.org/index.xml. AnimB Homepage.
13. T. Servat. Brama: A new graphic animation tool for B models. In J. Julliand and O. Kouchnarenko, editors, *B 2007*, LNCS 4355, pages 274–276, 2006.
14. S. Wieczorek, V. Kozyura, A. Roth, M. Leuschel, J. Bendisposto, D. Plagge, and I. Schieferdecker. Applying Model Checking to Generate Model-based Integration Tests from Choreography Models. In *Proceedings TESTCOM/FATES 2009*, LNCS 5826, pages 179–194. Springer-Verlag, 2009.