Separation of Considerations in Event-B Refinement toward Industrial Use

Naoto Sato[1,2], Fuyuki Ishikawa[1,3]

[1] The University of Electro-Communications, Japan
[2] Yokohama Research Laboratory, Hitachi Ltd., Japan
[3] National Institute of Informatics, Japan

Abstract. Formal method Event-B supports refinement as a means to divide a proof problem into different ones. To make the divided proofs easier to prove, we need to find an appropriate dividing strategy in refinement. At the same time, we should consider some other conditions given as proof obligations, and also how to formalize the specification. We think one of the reasons why Event-B is not accessible for developers is the simultaneity of these considerations in refinement. In this paper, we propose a modeling guideline to separate the consideration of the dividing strategy from Event-B formalism. We use a tree-structured diagram to design the dividing strategy in our guideline. We show a case study to confirm the feasibility of our approach.
Keywords: Event-B, refinement, modeling guideline

1 Introduction

Event-B is a well-established formal method for developing systems and proving their safety requirements. In industrial development, because specification description needs to be developed in natural language to agree with clients, we think Event-B can be useful to prove specification which has been developed in an informal way rather than to develop proved specification. However, Event-B is still unfamiliar to developers, specifically designers and architects. We think that's because Event-B is not so accessible for them. So, we should present Event-B in a more accessible way when we educate developers. Especially, we think *refinement* is one of the most difficult aspects to present to developers.

In Event-B refinement, we can express and prove a safety requirement on an abstract formal model of the system. Design details of the system are later gradually introduced into the formal models. During refinement, the proved safety requirement is preserved if and only if the abstract model simulates a concrete model, which is called *simulation constraint* in this paper. This means that refinement enables us to divide the proof on the concrete model into the proof on the abstract model and the simulation constraint. However, except for the simulation constraint, refinement doesn't provide any other guides or constraints on how to divide the proof problem by gradually refining the abstract model into concrete models. In this paper, we call a strategy to divide a proof problem *Dividing Strategy*. Moreover, we think modeling

specification in Event-B is not so trivial. We have several choices to formalize a given specification in Event-B semantics. It means we also have to consider how to formalize the specification during refinement.

Motivation. For the reason above, we have to consider the following three things at the same time in Event-B refinement: (1) how to follow the simulation constraint; (2) how many details of the specification should be introduced, as a dividing strategy; (3) how to formalize the specification in Event-B. Due to the difficulty of the simultaneous considerations, we think Event-B looks inaccessible for developers in learning. Accordingly, we are motivated to separate (2) the dividing strategy from (1) and (3) to reduce the difficulty. We also think the separation of the dividing strategy helps us motivate developers to learn Event-B. The separated dividing strategy makes it possible to share the outline of the proof that guarantees the correctness of the safety requirement with other developers and their clients.

Contribution. Our contribution in this paper is to propose an Event-B modeling guideline in which (2) the dividing strategy is separated from the others, aiming to reduce the difficulty of the simultaneous considerations in refinement for developers. In our guideline, we use *Dividing Strategy Tree (DST)* to design how to prove the safety requirement in the natural language apart from Event-B formalism. We suggest that the separation allows us to present Event-B to developers, namely non-experts of Event-B more accessibly. We also suggest that the separation motivates developers to learn and use Event-B because they can share the proof outline for consensus with other people unfamiliar with Event-B. In addition, to separate (3) from (1) within Event-B, we also propose that the most concrete model should be designed before refinement based on informally-developed specification.

Structure. The rest of this paper is structured as follows. In Section 2, we briefly review the Event-B modeling method. We motivate and present our approach in Section 3. We propose a modeling guideline with DST in Section 4. We provide an example developed according to our proposed guideline in Section 5. We discuss the benefits and limitations of our approach in Section 6. Related works are discussed in Section 7. Finally we draw conclusions and suggest future work in Section 8.

2 The Event-B Modeling Method

Event-B represents a further evolution of the classical B-method [2]. Event-B has a semantics based on transition systems and simulation between such systems. Event-B models are organized according to two constructs: *contexts* and *machines*.

Contexts. Contexts specify the static part of a model and may contain *carrier sets*, *constants*, *axioms* and *theorems*. Because we will not refer to contexts in this paper, we omit the details of them.

Machines. Machines specify behavioral properties of Event-B models. Machines may contain *variables*, *invariants*, *theorems* and *events*. Variables denoted by v define the state of a machine. They are constrained by invariants $I(v)$. Theorems are properties derivable from the invariants. Possible state changes are described by events. An event e can be represented by the term

$$e \triangleq \text{any } t \text{ where } G(t\,;\,v) \text{ then } S(t\,;\,v) \text{ end},$$

where *t* denotes the event's parameters, *G(t ; v)* is the event's *guard*, and *S(t ; v)* is the event's *action*. The guard states the condition under which an event may occur, and the action describes how the state variables are updated when the event occurs. The event's action is composed of one or more assignments of the form $x \coloneqq E(t ; v)$, where *x* is a variable in *v*, and *E* is an expression.

Refinement. Refinement provides a means to gradually introduce details of the specification into models [1]. A *concrete* machine can refine another *abstract* machine. At the moment, any behavior of the concrete machine should be simulated by a behavior of the abstract machine (*simulation constraint*). The simulation constraint is represented as logical proof assignments, which are called *proof obligations*.

Refinement can be done hierarchically, and a property proved in the abstract machine is then preserved in the concrete machines. Since the abstract machine contains fewer details than the concrete machines, the safety requirement in the abstract machine is usually easier to prove. Refinement enables us to divide a proof problem into different ones. As mentioned above, we call a strategy to divide a proof problem by multistep refinement *Dividing Strategy*.

3 Approach

To reduce the difficulty of the simultaneous considerations of (1), (2) and (3), we aim to separate (2) the dividing strategy from (1) and (3) in Event-B formalism. We introduce a modeling guideline using DST. DST has a hierarchical tree-structure of specification descriptions similar to Fault Tree [3] or KAOS goal model [4]. The safety requirement is put at the top node of the DST. We also put specification descriptions that achieve the safety requirement as children nodes. Subsequently, the child specification implements the parent at each branch (**Fig. 1**). We can use natural language to describe the specification in each node. We call a node of the tree *evidence* in this paper. Behaviors and properties of the system, as well as those of the environment can be written as evidences. Properties deduced from the specification are also described as evidences. We call a child evidence CE_i and a parent evidence PE_i for each branch *i* ($i \in [1..k]$). The parent evidence can be satisfied if all the child evidences are satisfied, that is, for *j* ($j \in [1..m]$), $(CE_{i1} \wedge ... \wedge CE_{ij} \wedge ... \wedge CE_{im}) \Rightarrow PE_i$.

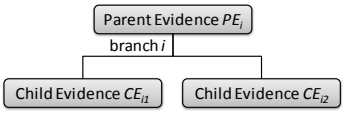

Fig. 1. Structure between parent evidence and child evidences in DST

Each branch *i* in DST also represents refinement step *i* in Event-B. Suppose that the abstract machine M_{i-1} is refined by the concrete machine M_i at refinement step *i*, we would design M_i so as to *meet* the child evidences CE_{ij} for all *j*. The definition of the statement *"a machine meets an evidence"* is given as follows.

Definition 1. An Event-B machine *M meets* an evidence *E* if and only if:

When E describes a property S, then M has an invariant representing S.
When E describes a behavior B, then M has events representing B.

There are several ways to design a machine so as to represent behaviors. When the evidence consists of pre-condition and post-condition of the behavior, we can introduce an event which has a guard as the pre-condition and an action as the post-condition. Or, when the behavior defines an execution order of events, we can introduce guard conditions into the events to control the order. To completely separate DST from the Event-B formalism, we don't define a formal notation of evidences or direct translation from evidences into machines. Because the parent evidence PE_i should also be met in M_i by refinement, any evidences which are met in M_i should be consistent with PE_i. Then, CE_{ij}s that achieve or implement PE_i in DST can come up for the evidences met in M_i. This means DST shows us how many details of the specification should be introduced in M_i, that is, the dividing strategy. Note that if M_i meets all CE_{ij}s, it may not satisfy the simulation constraint. Namely, the conjunction of CE_{ij}s is not a sufficient but necessary condition for the simulation constraint.

4 Modeling Guideline with Dividing Strategy Tree

In this section, we present the details of our proposed modeling guideline.
Conditions. We assume that the following two conditions are satisfied in the guideline: (Cond1) informal specification of the system and the environment has been developed, namely, it is given for Event-B modeling; (Cond2) the safety requirement is informally described as a property in the system specification.
Modeling Procedure. We now describe steps to build an Event-B model. We omit the design of the referred contexts in the procedure.

Step 1. Design the most concrete machine M_{k+1} that includes all the details of the given specification.
Step 2. Put the informal safety requirement as the top evidence in the DST.
Step 3. Design the machine M_0 in which the safety requirement is met.
Step 4. In the DST, choose a parent evidence PE_i which has no child evidence (called *leaf evidence*), and develop child evidences CE_{ij} which represent given specification or a property deduced from the specification. The child evidences CE_{ij} achieve or implement PE_i.
Step 5. Refine the machine M_{i-1} into the machine M_i. M_i should meet CE_{ij} for all j, and also satisfy the simulation constraint. If the simulation constraint can be proved, then go to Step 6. Otherwise, go back to Step 4 to update the DST.
Step 6. If there is a leaf evidence which represents a deduced property, then go to Step 4 for branch $i = i + 1$. Otherwise, go to Step 7.
Step 7. Refine the machine M_i into the most concrete machine M_{k+1}. If the simulation constraint can be proved, then terminate the procedure. Otherwise, go back to Step 4.

In our guideline, we also suggest that the most concrete machine M_{k+1} should be designed at the first step based on the informal given specification in order to separate the consideration (3) from (1). The formalization designed in M_{k+1} will be referred to in the following steps. In Step 2, the safety requirement that is informally given is put

as the top evidence. After designing the initial machine M_0 at Step 3, the DST is extended with child evidences CE_{ij} at Step 4. The extended branch is labeled by *i*. We refine the machine M_{i-1} into M_i by introducing CE_{ij}s at Step 5. If we can't design M_i which satisfies the simulation constraint, we will go back to Step 4 to update the DST. At the moment, unproved formulae in Step 5 can be referred to because the unproved formulae might represent missing or correct child evidences which will be met in M_i. In step 6, we check if all the leaf evidences represent given specification. The given specification is not dividable into child evidences anymore. In Step 7, we refine M_i. into the most concrete machine M_{k+1} with the remaining specification. The remaining specification is part of the given specification which is not introduced into the Event-B model yet. After finishing the procedure, we have the completed DST that shows "how the safety requirement was proved".

5 Example

To confirm the feasibility of our proposed guideline, we have two case studies on a simple file transfer protocol in [1] and switchover mechanism in a fault tolerant system. In this paper, we demonstrate our approach on the case study of the protocol.

Specification. As described in [1], the protocol is used by two agents. One agent called *sender* intends to transfer a sequential file denoted *f* to the other agent called *receiver*. The received file is denoted by *g*. The sequential file is transferred in *n* items called *blocks*. The most recent transferred block is denoted by *d*. The number of times of sending and receiving are denoted by *s* ($s \in [1..n+1]$) and *r* ($r \in [1..n+1]$). Accordingly, the block sent by the sender for the s^{th} time is denoted by *f(s)*. The same applies to *g(r)*. To alternate sending and receiving, the parity bits *p* and *q* are introduced. The bit *p* is inverted with every sending, and the bit *q* is inverted with every receiving.

Step 1. In this case study, we can see the most concrete machine in [1].

Step 2. The safety requirement for the protocol is also defined in [1] as follows.

Safety Requirement: After the transfer completes, *g* is equivalent to *f*.

We also put the requirement description at the top of the DST as shown in **Fig. 2**.

Step 3. The machine M_0 is designed to meet the invariant "inv0: $r=n+1 \Rightarrow g=f$" that represents the safety requirement. The event receive in M_0 is restricted to meet inv0 by the parameters p_g and p_r which denote the next values as follows.

receive ≙ any p_g, p_r where grd1:$p\_r=n+1 \Rightarrow p\_g=f$, grd2:$p\_g \in 1\cdots n \nrightarrow D$, grd3:$p\_r \in 1\cdots n+1$ then act1:$g \coloneqq p\_g$, act2:$r \coloneqq p\_r$ end

Step 4 for branch 1. The top evidence *E0* is chosen as a parent evidence. We develop the evidences *E1*, *E2* and *E3* as the child evidences that achieve *E0*.

Step 5 for branch 1. We refine M_0 into M_1 so as to meet the evidences *E1*, *E2* and *E3*. *E1* is represented as the invariant inv1 "$s=r \land r>1 \Rightarrow 1\cdots r-1 \lhd g = 1\cdots s-1 \lhd f$". The right side of the symbol "$\lhd$" is restricted to the left side. The evidence *E2* is represented by the guard grd1 of the event send as below.

send ≙ where grd1:$s = r$, grd2:$s \neq n + 1$ then act1:$s \coloneqq s + 1$ end

By grd1, the event send occurs only when s is equal to r. The guard grd2 is derived from $s \in [1..n+1]$. We also introduce a similar guard into the event receive for *E3*. Since M_1 designed this way is refined by M_0, we can prove the simulation constraint.

Step 6 for branch 1. Because there are the leaf evidences *E1*, *E2* and *E3* that are deduced from the given specification, we go back to Step 4 for branch 2.

Step 4-6 for branch 2. We develop the child evidences *E4*, *E5* and *E6* to implement *E1*. They are introduced into the refined machine M_2 as invariants. We choose *E2* as the next parent evidence, and then we go back to Step 4 to create the branch 3.

We could similarly design the machine M_3 to M_7. Finally, we found there is no leaf evidence which represents a deduced property. Then we moved to Step 7.

Step 7. We refine the machine M_7 into the most concrete machine M_8. Since we succeeded in the proof of the simulation constraint, we terminate the modeling.□

Finally, we completed the development of the Event-B model according to the proposed guideline. The completed DST is shown in **Fig. 2**. The proof statistics of our developments including the other case study is shown in **Table 1**.

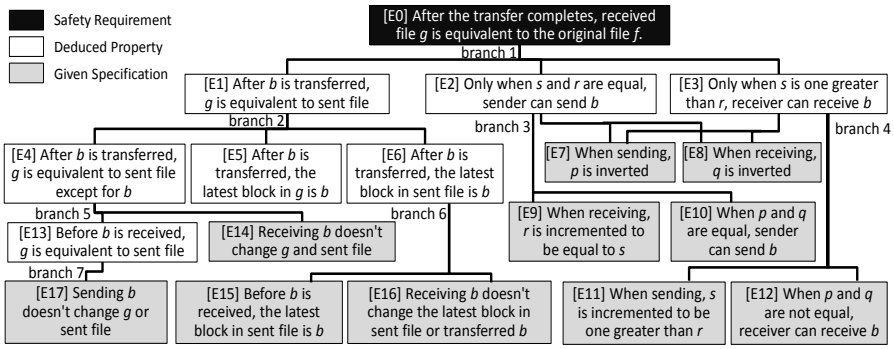

Fig. 2. Completed Dividing Strategy Tree

Table 1. Proof Statistics

#	Examples	Machines	Proof Obligations		
			Auto Proof	Interactive Proof	Total
1	A file transfer protocol	9	68	1	69
2	Switchover mechanism	11	114	7	121

6 Discussion

Although refinement is useful for proof, it's not always utilized because of its difficulty. In fact, in the industrial project Dependable Software Forum [13] to popularize Formal Methods in Japan, some people tried Event-B, but didn't use refinement. For those people, we aim to make refinement more accessible and motivating by the separation from Event-B formalism. In particular, we think it's important to present refinement more intuitively in a time-limited lecture or training course. So, we intend to extract the essential part of refinement for developers, which is "how the safety requirement was proved". This encourages them to understand what refinement is, and what impact refinement has on their development. We think one can say the same

thing in other formal methods. For example, in a model checking method, the abstraction of the specification should be considered to avoid the state explosion at the same time as the formalization. Similar separations may be effective in the education.

Because one has to follow only the simulation constraint in Event-B refinement, there are various ways to refine the system. This could be an advantage for experts of Event-B because they are allowed to develop the model as they like. On the other hand, that could be a disadvantage for non-experts. In our guideline, we restrict the modeling approach to *top-down* by DST. Top-down means to start modeling from the whole system and divide it into its components. It's also called *outside-in* approach in [10], and has been applied in a number of case studies [11] [12]. We think this restriction is helpful for non-experts because the restriction makes how to start refinement clearer. However, our guideline is not effective for systems which are not suitable for the top-down approach. Another limitation is that our guideline only aims to *separate* the considerations, but does *not solve* the difficulties of the considerations (1), (2) and (3) themselves. This means developers who model the system according to our guideline should be capable of offering solutions for the considerations.

7 Related Work

There are a number of approaches to take advantage of tree-structured diagrams in Event-B refinement. In [5], the authors provide a method in which KAOS goal models [4] are translated into Event-B models. The authors of [7] also propose an approach to utilize a tree-structured diagram based on KAOS for Event-B refinement. These methods are similar to our approach in that the parent goal is translated into the abstract machine, and its child goals are translated into the concrete machine. However, because they aim for direct translation from goals to machines, the descriptions of goals are limited to semantically correspond to events of Event-B. This means that developers have to consider not only (3) but also (1) when they develop the goal models, even though these considerations are done in the natural language.

In contrast, we don't define how to describe evidences in DST, or translation into Event-B in our approach. This enables us to completely separate (2) the dividing strategy from the others. In general, the difference between [5], [7] and our approach is how to split the assigned consideration (1), (2) and (3). It's difficult to clearly say which way is generally better because the amount of considerations is the same either way. However, we think at least that our approach must be more accessible for non-experts because (2) the dividing strategy is completely independent from Event-B formalism. Moreover, our approach is better to share the dividing strategy as a sketch of the refinement with other people who are not familiar with Event-B.

Another group in [9] works on the traceability between KAOS and Event-B. Since they only use the leaf goals, goal refinement in KAOS doesn't correspond to machine refinement in Event-B, unlike [5], [7] and our approach.

From a more general perspective of refinement support, the work of [6] proposes a Refinement Planning Sheet to overview refinement. Unlike our approach, it doesn't intend the separation from Event-B formalism. The authors in [8] also provide an

approach to generate a refinement plan which has the highest evaluation according to their criteria. However, the generated strategy is not always appropriate. For this reason, we do not remove, but rather separate, (2) the dividing strategy in our approach.

8 Conclusion

In this paper, we presented a modeling guideline to make refinement more accessible and motivating for developers in learning Event-B. Our approach allows them to separate (2) the dividing strategy by DST from (1) the simulation constraint and (3) the formalization in Event-B. We did two case studies, one of which was presented in this paper. Based on those results, we confirmed the feasibility of our guideline.

As future work, we plan to ask developers to try our guideline. We would need to evaluate how accessible and motivating our guideline is in practice in terms of how much time and stress there is in modeling. In addition, comparative evaluation with other related approaches referred to in Section 6 is also important.

References

1. J.-R. Abrial, Modeling in Event-B: System and Software Engineering, Cambridge Univ. Press, 2010.
2. J.-R. Abrial, The B-book: Assigning Programs to Meanings, Cambridge Univ. Press, 1996.
3. W. E. Vesely, F. F. Goldberg, N. H. Roberts, and D. F. Haasl, Fault Tree Handbook (NUREG-0492), U.S. Nuclear Regulatory Commission, Washington, D.C., January 1981.
4. A. van Lamsweerde, Requirements Engineering: From System Goals to UML Models to Software Specifications. Wiley, 2009.
5. A. Matoussi, F. Gervais, and R. Laleau, A goal-based approach to guide the design of an abstract Event-B specification. In Engineering of Complex Computer Systems (ICECCS), 2011 16th IEEE International Conference on, pages 139–148. IEEE, 2011.
6. S. Nakajima, A Refinement Planning Sheet, Rodin User and Developer Workshop, University of Duesseldorf, 20-22 September 2010.
7. K. Traichaiyaporn, T. Aoki, Refinement Tree and Its Patterns: A Graphical Approach for Event-B Modeling Formal Techniques for Safety-Critical Systems, Communications in Computer and Information Science, Volume 419, 2014, pp 246-261
8. T. Kobayashi, S. Honiden, Towards Refinement Strategy Planning for Event-B, Proceedings of DS-Event-B 2012: Workshop on the experience of and advances in developing dependable systems in Event-B, Nov. 2012.
9. C. Ponsard and X. Devroey, Generating high-level Event-B system models from KAOS requirements models. In Actes du XXIIème Congrès INFORSID, France, 2011.
10. S. Hudon and T.S. Hoang, Development of control systems guided by models of their environment, Electronic Notes in Theoretical Computer Science, vol.280, pp.57-68, 2011.
11. DEPLOY Deliverable D16, D2.1 Pilot Deployment in Transportation (WP2), 2009.
12. P. Inna, T. Elena, and L. Linas, Development of Fault Tolerant MAS with Cooperative Error Recovery by Refinement in Event-B, Proceedings of DS-Event-B 2012: Workshop on the experience of and advances in developing dependable systems in Event-B, Nov. 2012.
13. Information-technology Promotion Agency, Reports on Dependable Software Forum, 2012, http://www.ipa.go.jp/sec/softwareengineering/reports/20120928.html (in Japanese)