Timing diagrams add Requirements Engineering capability to Event-B Formal Development

Tossaporn Joochim and Michael R. Poppleton
Dependable System and Software Engineering Group
School of Electronics and computer Science
University of Southampton, United Kingdom
{tj04r,mrp}@ecs.soton.ac.uk

Abstract

Event-B is a language for the formal development of reactive systems. At present the RODIN toolkit [15] for Event-B is used for modeling requirements, specifying refinements and doing verification. In order to extend graphical requirements modeling capability into the real-time domain, where timing constraints are essential, we propose a Timing diagram (TD) [13] notation for Event-B. The UML 2.0 based notation provides an intuitive graphical specification capability for timing constraints and causal dependencies between system events. A translation scheme to Event-B is proposed and presented. Support for model refinement is provided. A partial case study is used to demonstrate the translation in practice.

1. Introduction

There are many ways to describe the system requirements such as using goal orientation e.g. Knowledge Acquisition in Automated Specification (KAOS), Formal methods (FM) such as Event-B, and graphical notations such as UML diagrams.

KAOS [3] is a Goal-modeling requirement specifications technique. KAOS uses *Goal* and *Operational models* to declare system requirements in a form of Linear Temporal Logic (LTL), and has a concept of goal refinement to refine a goal into subgoals. Event-B [8, 17] is a formal method (FM) that describes system requirements in a form of a set-theoretic notation. Event-B is used to improve formal requirements analysis, verifying each next level of detail (refinement) and helping eliminate error early in the design process. The timing diagram (TD) is an UML2.0 notation, used to show the behaviors of objects over time.

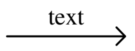
As [12, 14, 16], FM can be difficult to construct and demands trained professionals. Demonstration of requirements in graphical forms helps software developers to define specification more easily than by using the FM mathematical notations. Thus, there are many researchers trying to bridge the gap between Event-B and UML diagrams. For example [1, 7, 9] and UML-B [5, 6]. UML-B is a front end to the Event-B and is a toolkit developed in RODIN [15]. The tool provides graphical modeling capability in term of UML-like Class diagrams and Statecharts. Even though those researchers combining Event-B with UML diagrams, do not focus on the combination of timing constraints and casual dependencies among different objects' states, other essential part of requirements analysis, to the Event-B model as our research does.

The aim of our research is to develop techniques to specify requirements by using graphical notations, i.e. TD, and then generating KAOS, Event-B and UML-B models. Our work comprised two main steps. First, we amend UML2.0 TD notation and define its Backus Naur Form (BNF) [4]. Second, we generate three kinds of pattern to create KAOS, Event-B and UML-B models from TD. A lift system based on Jackson's work [10] is used as a primitive case study. The case study has been modified by adding timing constraints to dependency requirements on events to demonstrate the issues. In this paper, we present a subset of our complete set of translation rules for generating Event-B from TD. The translation rules from TD to KAOS can be found in [18].

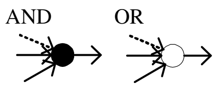
In section 2, we describe the TD; section 3 illustrates case study specifications; section 4 describes the Event-B modeling; section 5 explains how patterns are used for generating Event-B; section 6, we make conclusions.

2. Timing Diagrams

Examples of TD notations are used in the research is shown below



An arrowed line indicates cause and effect between objects. The beginning of line represents the cause while the end of the line (with arrow) represents the effect. One can identify additional conditions which make state changes by plain text above the arrowed line. A duration constraint is used to describe how long a state or value must be in effect and identified by a symbol [t1,t2] where t1 and t2 indicate the timing constraint starts from t1 to t2.



"AND" and "OR" notations are used for specifying relationships within between *CauseEffect arrows* (note, they are not used to contribute a cause to many effect segments). A node can has many "AND" or "OR" causes which are represented by dash-line here.

3. Case study

Figure 1 shows part of the lift specification used in this paper. The requirements maybe described "...A part of the lift system contains two objects: the floor sensor and the lift. The lift movement states are separated into three steps: moving up, stop at floor and moving down. The floor sensor has two states: on and off. The relation between the lift movement and the floor sensors means whenever a user presses a button to request the lift, the lift starts moving up/down (1 and 2) from the current floor; within between 2 – 5 seconds after the lift starts moving, the floor sensor of the current floor will turn off.…"

In Figure 1, in term of TD notations, we can describe that there are two *Timelines* which represent the state change in time for that particular object: *floorsensor* and *lift*. The lines 1 and 2 show the combination of the *CauseEffect arrow* by using "*OR*" notation; it means the *floorsensor* is set to *Off* according to whether the *lift* is in the state of *MvgUp* or *MvgDwn*. Predicates such as $f = currentFl \wedge dir = Up$ are additional conditions on the *CauseEffect arrow* where f represents a floor and is a dynamic state parameter that can change in time. Object states and their indices such as *On1* and *Off2* represent segments of the *floorsensor Timeline*. They are not TD notations but used in translation rules as described in section 5.

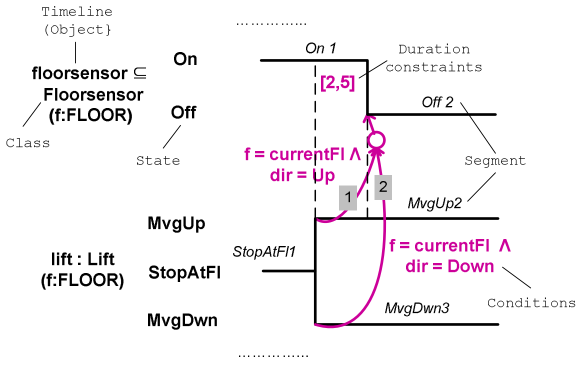


Figure 1. Timing diagram

4. The Event-B Modeling

The dynamic part of an Event-B model is called the ***MACHINE.*** The ***MACHINE*** includes state ***VARIABLES*** definitions, an ***INVARIANT*** predicate on the state, ***INITALISATION*** and ***EVENTS***. In Event-B, the units of behavior are called ***EVENTS***. Each event *E* is composed of an enabling guard *G(l,v)* and an action *S(l,v)* where *v* are state variables constrained by invariants *I(v)*; *l* are local variables that the event may contain. The general form of an event is, *E* = ***ANY*** *l* ***WHERE*** *G(l,v)* ***THEN*** *S(l,v)* ***END.*** The structure of a Event-B model is shown below

MACHINE *name* =
    VARIABLES *v*, …..   ***INVARIANT*** *I(v)*, …..
    INITIALISATION …..
    EVENTS
        eventname = ***ANY*** …***WHERE*** … ***THEN*** … ***END***
        eventname = ……
    END

5. Pattern Transforming Timing Diagrams into Event-B

5.1 BNF Timing Diagrams definitions

We identify TD BNF definitions and use them to create the translation rules to transform TD to Event-B. Examples of BNF definitions for *Timeline* are shown in the following.

A *Timeline* comprises a chain of segments which individual segment represents the object state (`Objst`) and its position (`Index`) in the *Timeline*.

```
Timeline ::= Segment+
Segment ::= Objst Index; Index ::= INT
```

5.2 Translation rules

Normally, one rule comprises many basic sub-rules. Examples of those basic sub-rules are illustrated below

Rule 1: **TObj**(Segment) $\rightarrow$ Obj;

This rule gives the object for an input segment.

Rule 2 : **TState** (Segment) $\rightarrow$ Objst;

This rule gives the object state for an input segment.

In this paper we demonstrate **TTimeCtrnt** rule as an example for generating a duration constraint as guards for *foorsensorOff* event (figure 2). This rule is defined as recursion from the main rule (not shown in this paper) and uses a segment (segm) as an input parameter. The detail of the rule is shown in the following.

```
TTimeCtrnt(segm) →
  (gclock – TObj(segm)TState(segm)Time)≥
      LOWER_LIMIT_TObj(segm)  ∧
  (gclock – TObj(segm)TState(segm)Time)  ≤
      UPPER_LIMIT_TObj(segm)
```

When *MvgUp2* and *MvgDwn3* are used as parameters, parts of event's guards are generated as shown in figure 2 (in rectangle); where *LOWER_LIMIT_floorsensor = 2* and *UPPER_LIMIT_floorsensor* = 5 are created as constants by another rule (not show in this paper).

```
foorsensorOff =
  ANY….
  WHERE  .....
 ( ((gclock – liftMvgUpTime) ≥
       LOWER_LIMIT_lift  ∧                 From
   ( gclock – liftMvgUpTime) ≤            MvgUp2
       UPPER_LIMIT_ lift  ))   ∨

  (  (gclock – liftMvgDwnTime) ≥
       LOWER_LIMIT_ lift  ∧                From
   ( gclock – liftMvgDwnTime) ≤           MvgDwn3
       UPPER_LIMIT_ lift)) )
  THEN  .....
  END
```

Figure 2. floorsensorOff event

6. Conclusions

We use TD to represent system requirements and then transform the TD into KAOS, Event-B and UML-B. The contributions of our research are described in the following.

1. Modeling : Even though the information on TD can be expressed in other diagrams such as using Statecharts [5,6,7,9], it is not in a helpful way for the users. For example, one can put timing constraints and state conditions into Statecharts but one Statecharts refers to other Statecharts for the dependency. If the Statecharts have guards related to other Statecharts, then we have guards on the state transitions here which refer for something going on somewhere else. The causal interaction between the objects cannot be seen on one diagram of view. Thus, we have many diagrams in the same time which it is hard to read. It is not helpful for the users in term of modeling. In TD, we can describe the causality explicitly in the arrows between events and have them all in the same diagrams. The TD notations include graphically described timing constraints. It is very natural to form timing constraints expression in TD. Thus, our contribution is adding this single-view modeling capability to Event-B.

2. Formal requirements : Event-B is a well known method using in critical systems because it has techniques of formal proof and model checking of correctness properties. However, as in section 1, Event-B is claimed to be difficult to construct. Thus, we propose TD to capture the formal requirements and provide methodology to transform TD into Event-B and UML-B. This is helpful for users in term of identifying formal requirements by using graphical notations rather than mathematics notations.

3. KAOS lacks concepts of proof obligations (POs) as in Event-B. However, it is a semi-formal method due to using LTL to identify Goal and Operational models; TD can describe some LTL operators such as *X* (next) over some period of time. Then, our contribution is to combine graphical notations, TD, with semi-formal method, KAOS (have done), and then aim to generate to FM, Event-B (have not done). Finally, the output can be proof by Event-B toolkit for the correctness.

There is a limitation with TD, that is, it is not designed to add state-based information. Currently, we are creating a pattern to generate UML-B from TD using Atlas Transformation Language [2]. The integration with UML-B is beneficial for TD because one can add extra information that is missing from TD by using UML-B. To verify the correctness of Event-B model, we use RODIN Event-B toolkit [15] and the ProB [11] tool for syntax checking, proof obligations, animation and model-checking.

7. References

[1] A. B. Younes, and L. J. Ayed, "B.: Using UML Activity Diagrams and Event B for Distributed and Parallel Applications" *In: 31st Annual International Computer Software and Applications Conference*, Vol.1, COMPSAC, 2007, pp. 163-170.

[2]ATL, "ATLAS Transformation Language", http://www.eclipse.org/m2m/atl/

[3] E. Letier, and A. van Lamsweerde, "Agent-Based Tactics for Goal-Oritented Requirement Elaboration", *In: ICSE 2002 - 24th International Conference on Software Engineering"*, ACM Press, 2002, pp. 83-93.

[4]C. Métayer, J.-R. Abrial, and L. Voisin, "Event B Language", 2005, http://rodin.cs.ncl.ac.uk/deliverables/D7.pdf

[5] C. Snook, and M. Butler, "UML-B : Formal modeling and design aided by UML", *In: ACM Transactions on Software Engineering and Methodology*, 2006, pp. 92-122.

[6] C. Snook, and M. Waldén, "Refinement of Statemachines using Event B Semantics", *In: B 2007: Formal Specification and Development in B, 7th International Conference of B*, 2007.

[7] H. Ledang, and J. Souquieres, "Contributions for Modelling UML State-Charts in B", *In: Third International Conference on Integrated Formal Methods*, 2002.

[8] J.-R. Abrial, and S. Hallerstede, "Refinement, Decomposition and Instantiation of Discrete Models: Application to Event-B", Findamentae Informatica, 2006.

[9] L. Jiufu, "Integration of statechart and B method based analysis and verification for flight control software of unmanned aerial vehicle" *ACM SIGSOFT Software Engineering Notes*, Vol. 32, Issue 2, 2007, pp. 1-4.

[10] M. Jackson, *Problem Frames Analysis and Structuring Software development problems*, Addison-Wesley, 2001, pp. 48-75, 177-91.

[11] M. Leuschel, and M. Butler, "The ProB Animator and Model Checker for B: A Tool Description", *In: 12th International FME Symposium 2003 (FM2003)*, Italy, 2003, pp. 855-74.

[12] N. Bashar, and S. EasterBrook, "Requirement Engineering: A Roadmap", *In: Conference on the The Future of Software Engineering,* Ireland, 2000.

[13] OMG, "UML Superstructure Specification v2.0", http://www.omg.org/cgi-bin/doc?formal/05-07-04.

[14] P. J. Bowen, and G. M. Hinchey, "Ten Commandments of Formal Methods…Then Years Later", *IEEE Computer Society*, Vol. 39, No. 1, Jan, 2006, pp. 40 – 48.

[15] Rigorous Open Development Environment for Complex Systems (RODIN) – IST 511599, http://rodin.cs.ncl.ac.uk/

[16] R. Razili, C. Snook, M. Poppleton, P. Garratt, and R. Walters, "Experimental Comparison of the Comprehensibility of a UML-based Formal Specification versus a Textual One", *In: 11th International Conference on Evaluation and Assessment in Software Engineering (EASE'07)*, UK, April, 2007.

[17] S. Hallerstede, "Justifications for the Event-B Modelling Notation", *Lecture Notes in Computer Science*, Springer, 2006, pp. 49-63.

[18] T. Joochim, and M. Poppleton, "Transforming Timing Diagrams into Knowledge Acquisition in Automated Specification", *In: IAIT2007: The 2nd International Conference on Advance in Information Technology*, Thailand, 2007.

Expectation and benefits statements

1. How can Timing diagram be used to advance the requirement engineering and Formal method?

2. How is the beneficial of using Timing diagrams to model critical systems' requirements?

It would be helpful for us to justify whether the Timing diagram is useful for use in the critical systems. Moreover, we would like to have opinions/suggestions for what kind of state-based system that cannot be explained by Timing diagrams? So, we can identify the limitations of our Timing diagram.

3. How far the Timing diagram can be used to demonstrate human actions? How can we model environments of the system if that is human?

Since there are many requirements concern with human activities, for example in the lift system that needs human to request the lift by pressing buttons. In this case, we can demonstrate the pressing activity by represent it as an event in Event-B.

However, there is a case study such as Ambulance Service system. The timing constraints are concerning with responding to emergency calls requiring the rapid intervention of an ambulance. How can we identify these requirements in the Timing diagrams and how we control the timing constraints?

4. Is there any issues/problems with our currently translation rules?

We really appreciate if anyone can point out any problems in our model. So, we can correct the translation rule in order to obtain complete models.

5. What are the model integration issues we should be aware of?