Using formal methods in distributed system design

Awwama Emad, *Kadi* Mohammad, *Krayem* Said, *Lazar* Ivo, *Rihawi* Ahmad

University UTB, FAI department, Nad Stráněmi 4511, 760 05 Zlín, Czech Republic

Abstract. Distributed systems are groups of networked computers, which have the same goal for their work. The terms "concurrent computing", "parallel computing", and "distributed computing" have a lot of overlap, and no clear distinction exists between them. The same system may be characterized both as "parallel" and "distributed"; the processors in a typical distributed system run concurrently in parallel. Parallel computing may be seen as a particular tightly coupled form of distributed computing, and distributed computing may be seen as a loosely coupled form of parallel computing. Nevertheless, it is possible to roughly classify concurrent systems as "parallel" or "distributed" using the following criteria. Philosophy is centrally concerned with arguments. The first question to be asked of any argument (or inference) is whether or not it is valid: that is, does its conclusion really follow from the cited premises? Validity of inference is the central problem of deductive logic.

1 Introduction

Recently, with the increasing demand benefit of using formal methods for modeling, we started seeing large number of applications using formal methods Examples of these methods include ASM (Borger & Stark, 2003), B (Abrial, 1996), and VDM (Jones, 1990). We have choosed Event-B as a formal method to show effectiveness of this method, which has possibility for developing system free of errors by verifying using Rodin model.

Formal verification of a program is the mathematical proof that it does what is expected of it. The 21st century has seen a vast worldwide interest in formal methods [1-5].

While Rigorous descriptions promise to improve system reliability, design time and comprehensibility, they do so at the cost of an increased learning curve; the mathematical disciplines used to formally describe computational systems are outside the domain of a traditional engineering education. In addition, the meta-models used by most formal methods are often limited in order to enhance provability. There is a notable tradeoff between the need for rigor and the ability to model all behaviors.

1.1 Event-B method

1.1.1 The Event-B formalism

We present our formal development framework — Event-B (see Figure 1).

The Event-B formalism is a state-based formal approach that promotes the correct-by-construction development paradigm and formal verification by theorem proving.

Event-B has been specifically designed to model and reason about parallel, distributed and reactive systems.
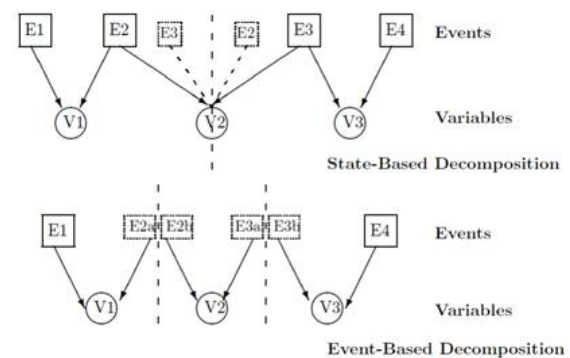
Fig. 1. Comparing event-based and state-based decomposition.

Formal verification involves the application of mathematical proofs to every possible behavior allowed by a specification (Abrial, 1996). In a state-based specification the behavior is a transformation of the system moving from one state to another. Proof obligations are generated using the specification and the language rules

These proof obligations then need to be discharged using properties of the specification.

Event-B is a mathematical approach for developing formal models of systems (Abrial & Hallerstede, 2006) [11, 7]. An Event-B model is constructed from a collection of modelling elements. These elements include invariants, events, guards and actions. The modelling elements have attributes that can be based on set theory and predicate logic. Set theory is used to

---

* Corresponding author: lazar@fmk.utb.cz

represent data types and the manipulation of data. Logic is used to apply conditions on the data.

1.1.2 The development of an Event-B model

The development of an Event-B model goes through two stages; abstraction and refinement. The abstract machine specifies the initial requirements of the system0 Refinement is carried out in several steps with each step adding more detail to the system, generally, but not exclusively, in a top-down manner Reactive systems (Harel & Pnueli, 1985) are systems that continually respond to changes in their environment. The focus on atomic events in Event-B creates a representation of a reactive system (Jones, 2005). The model transitions are triggered by changes in the state of the model, which can represent the system's environment. The guard on an event will allow or prevent an event from occurring depending on the state of the model. When none of the guards are true the system is deadlocked. Event-B is designed for modelling distributed systems (Abrial & Hallerstede, 2006). It implements the theory of discrete transition systems. Discrete transition systems, or action systems, model atomic actions that can be performed in parallel providing the actions do not affect the same state variables One method for specifying concurrency in Event-B is to model each update as a group of potentially interleaving atomic events (Edmunds & Butler, 2008) [1-5].

This allows the model to specify how concurrent execution can be dealt with by the system being modelled specifying a distributed system in Event-B takes a global approach. Rather than creating a specification for each component of the system it is modelled as a whole along with its environment. The model is closed in that it reacts only to changes in its internal state. Initially states are modelled abstractly with the events that describes the main goal of the system.
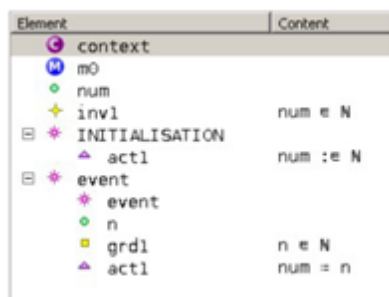

Fig. 2. Context Event-B.

Detail is added through refinement to describe the final distributed system. The ability to add new events and refine single events into multiple concrete events allows the functionality of the system to expand beyond that modelled in the abstract machine. Refinement ensures that the refined models are consistent with the abstract machine.

1.2 Event-B model

In Event-B, a system model is specified using the notion of an abstract state machine (Abrial, 2010) [6,8]. An abstract state machine encapsulates the model state represented as a collection of model variables, and defines operations on the state, i.e. it describes the dynamic behavior of the modelled system. A machine may also have the accompanying component, called context (see Figure 2). A context might include user-defined carrier sets, constants and their properties, which are given as a list of model axioms. In Event-B, the model variables are strongly typed by the constraining predicates called invariants. Moreover, the invariants specify important properties that should be preserved during the system execution. A general form of Event-B models is given in (see Figure 3).

The machine is uniquely identified by its name M. The state variables, v are declared in the Variables clause and initialised in the Init event. The variables are strongly typed by the constraining predicates. I given in the Invariants clause. The invariant clause might also contain other predicates defining properties that should be preserved during system execution.

The dynamic behavior of the system is defined by the set of atomic events specified in the Events clause. Generally, an event can be defined as follows introduction to the event-B method and the Rodin [12, 13].

We outline the general structure of an Event-B specification. A specification consists of a static part, specified in a context, and a dynamic part, specified in a machine.

An Event-B machine M1 may be declared to be a refinement of some other Event-B machine M0. In this case we refer to M0 as the abstract machine and M1 as the refined machine. Machine M1 is said to be a correct refinement of M0 if any behavior that may be exhibited by M1 is also a possible behavior of M0. Refinement represents our expectation that the behavior of M1 should conform to the behavior of M0. Of course declaring that M1 refines M0 does not on its own guarantee the correctness of a refinement. Rather the declaration gives rise to proof obligations that need to be discharged in order to guarantee the correctness of a refinement.
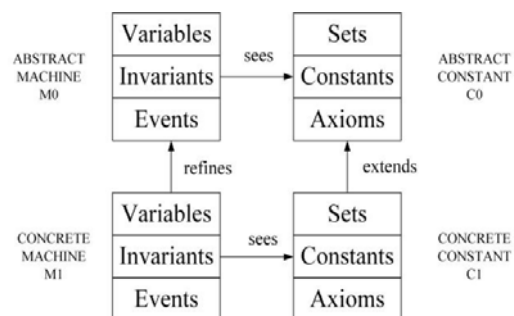

Fig. 3. Machine and context relationship.

When refining a machine, it is common to specify new types and constants to be used in the refinement.

This is achieved by specifying a new context for the refined machine. If the specification of any new types and constants depend on the types and constants used by the abstract machine, the new context is declared to be an extension of the context of the abstract model. The relationships between a machine and its refinement, as well as their respective contexts, is illustrated by Figure 2. This figure shows the refinement declaration from M1 to M0, together with the relationships with their contexts. A refined context C1 is declared as an extension of the abstract context C0 meaning context C1 may refer to types and constants specified in context C0. The dashed line from machine M1 to context C0 indicates that M1 implicitly see definitions in C0 (via C1).

An Event-B context contains the following elements:

- Sets: Abstract types used in specification to distinguish various entities
- Constants: Logical variables whose value remain constant
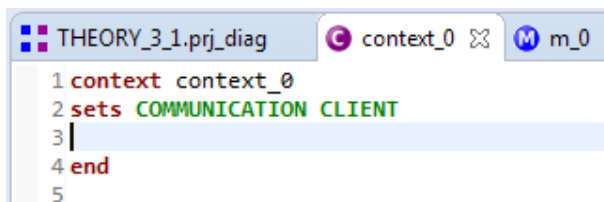- Axioms: Predicates that specify assumptions about the constants.

An Event-B machine contains the following elements:

- Variables: State variables whose values can change
- Invariants: Predicates that specify properties about the variable that should always, remain true.
- Initialization: Initial values for the abstract variables
- Events: Guarded actions specifying ways in which the variables can change. Events may have parameters.

A machine may see the static elements defined in a context meaning that these elements are visible within the machine. The structure of a specification is outlined (see Figure 3).

2 Our proposed model

Our model represent a client management model verified using Rodin too.

Fig. 4. Context in the machine m_0. Case study I — platform Rodin.

3 Conclusions

Communication and negotiation are very important characteristics of distributed systems; and in this paper we have presented some of the basic concepts in formal method using Event-B also we have presented verification of protocols in distributed systems., so we can say, event-B allows us to define a kind of modeling methodology by write the correct mathematical notions;

wherefore we can apply event-B in modeling many different complex projects, but we should choose carefully invariants and variables to ease effort of proof.

As well as the Rodin tool offers reactive environment for constructing and analyzing models as do most modern integrated development environments, and provides integration between modeling and proving whereas this is important feature for the developers to focus on the modeling task without switch between different tools to check proving in same time.

The intent of this paper to give some insights on modelling and formal reasoning using Event-B method in distributing systems.

References

1. E. Borger, R. Stark *Abstract State Machines: A Method for High-Level System Design and Analysis*. Springer, ISBN 978-3-642-18216-7. (2003)
2. C.B Jones. *Systematic software development using VDM*. New York: Prentice Hall ISBN:0-13-880733-7, **2** (1990)
3. B. Meyer, B. In *Advances in Object-Oriented Software Engineering*, D. Mandrioli and B. Meyers, Eds. Prentice Hall, 1-50. (1991).
4. P. Behm, P. Benoit, A. Faivre, & Meynadier, J.-M. météor: A successful application of B in a large project. In *World Congress on Formal Methods*, J. M. Wings, J. Woodcock, & J. Davies, Eds. Lecture Notes in Computer Science, vol. 1708. Springer, Berlin, Heidelberg, 369–387. ISBN:3-540-66587-0, **1** (1999)
5. I. Houston, S. King, Experiences and results from the use of Z in IBM. In VDM '91: *Formal Software Development Methods.* Lecture Notes in Computer Science, vol **551**. Springer, Berlin, Heidelberg, 558–595. (1991).
6. J.-R. Abrial. *Modeling in Event-B*, Swiss Federal University (ETH), Zürich, 2010. ISBN: 978-0-521-89556-9
7. J.-R. Abrial, M. Butler, S. Hallerstede, & L. Voisin An open extensible tool environment for Event-B, *Springer*, of*LNCS*, 2006, pp 588–605, ISSN 0302-9743. Vol. **4260** (2006)
8. J.-R. Abrial. *Modeling in Event-B*, Cambridge, ISBN-13 978-0-511-72976-8, CAMBRIDGE UNIVERSITY PRESS (2010)
9. M. Jastram, M. Butler. *Rodin User's Handbook*: Covers Rodin v.2.8, CreateSpace Independent Publishing Platform, ISBN 10: 1495438147 ISBN 13: 9781495438141, USA. https://www3.hhu.de/stups/handbook/rodin/current/pdf/rodin-doc.pdf. (2014)
10. Rodin User's Handbook v. 2.8, Accessed on: 2014-02-04 https://www3.hhu.de/stups/handbook/rodin/current/html/introduction.html.

11. Hallerstede, S. *Justifications for the Event-B Modelling Notation*. B2007: Formal Specification and Developmentin B, Springer, Vol. **4355**., pp 49-63., ISSN 0302-9743. (2007)

12. http://wiki.event-b.org/index.php/Rodin_Platform_3.2_Release_Notes*Rodin Platform 3.2 Release Notes*, Accessed on: 2015-11-05

13. http://www.event-b.org

```
 1 machine m_0 sees context_0
 2
 3 variables replyed choosed fulfilled declined cs
 4
 5 invariants
 6   @inv1 cs∈COMMUNICATION ⇸ CLIENT ∧ replyed ⊆ cs ∧ choosed ⊆ replyed ∧ fulfilled ⊆cs  ∧ declined ⊆ cs
 7
 8 events
 9   event INITIALISATION
10     then
11       @act1 replyed = ∅
12       @act2 choosed = ∅
13       @act3 fulfilled = ∅
14       @act4 declined = ∅
15       @act5 cs = ∅
16   end
```

Fig. 5. Invariants and initialization in the machine m_0. Case study I — platform Rodin.

```
18   event Plan_in
19     any c as
20     where
21       @grd1 c ∈ COMMUNICATION  ∧ c ∉ dom(cs) ∧ as ∈ COMMUNICATION⇸CLIENT ∧ dom(as)={c} ∧ ran(as)⊆CLIENT
22     then
23       @act1 cs=cs  ∪ as
24   end
25
26   event choose
27     any c as
28     where
29       @grd1 c∈ dom(replyed) ∧ as ⊆ replyed ∧ as ∩ choosed =∅
30     then
31       @act1 choosed = choosed ∪ as
32   end
33
34   event decline
35     any c as
36     where
37       @grd1 c ∈ dom(cs) ∧ c ∉ dom(declined) ∧ as ⊆ {c} ◁ cs
38     then
39       @act1 declined = declined ∪ as
40   end
41
42   event reply
43     any c a
44     where
45       @grd1 c ↦a ∈ cs ∧ c ↦a ∉ replyed
46     then
47       @act1 replyed = replyed ∪ {c↦a}
48   end
49
50   event fulfill
51     any c as
52     where
53       @grd1 c∈dom(choosed) ∧ c ∉ dom(fulfilled) ∧ as ⊆ {c} ◁choosed
54     then
55       @act1 fulfilled = fulfilled ∪ as
56   end
57 end
```

Fig. 6. Events in the machine m_0. Case study I — platform Rodin.