Distributed System Model Using SysML and Event-B

Qi Zhang(✉), Zhiqiu Huang, and Jian Xie

Nanjing University of Aeronautics and Astronautics,
Nanjing, People's Republic of China
zhangl993@nuaa.edu.com,
{zqhuang,xiejian_5}@nuaa.edu.cn

Abstract. Distributed system is more complicated compared with other systems due to its concurrency and distribution. Moreover, the traditional system development process is usually informal, and a large number of tests are required. On the other hand, the formal methods have been applied in many system development fields and many achievements have been made. In this paper, a method which combines SysML requirement diagrams and Event-B to model distributed system is proposed, including their mapping relations.

Keywords: Distributed system · SysML · Event-B · Requirement diagram

1 Introduction

Compared with the traditional centralized system, the distributed system is more complicated due to its concurrency and distribution. Though the distributed system has developed rapidly with kinds of specifications and standards in recent years, there still exists some shortcomings. Because these specifications and standards usually lack solid theoretical foundation, it's difficult to give a formal specification of distributed systems as well as the correctness verification. As the distributed system becomes increasingly complicated, the formal methods are needed to help overcome these shortcomings in development.

The formal methods are used to help model complex system in a mathematic way [10]. In formal development method, the text-based requirements are formalized, and with the help of formal developing tools, hazards and errors can be automatically detected. Event-B is a formal specification language for modeling and verifying system requirements [1]. The basic idea that distinguishes Event-B from other formal methods is its refinement mechanism. In Event-B refinement process, the abstract specifications can be transformed into concrete specifications gradually until all requirements in the specification contained in the model. Besides, a set of proof obligations (POs) are generated after every refinement stage, which are used to verify the consistency from its abstract model and to make sure that the functional requirements have been correctly added [9].

X. Gu et al. (Eds.): MLICOM 2017, Part I, LNICST 226, pp. 326–336, 2018.
https://doi.org/10.1007/978-3-319-73564-1_32

Although Event-B provides a refinement mechanism to gradually model the system, one of the major problem is that there is no standard guideline to use the refinement mechanism. When modeling complex systems, it is difficult for developers to organize the refinement steps. On the other hand, the main weakness of using formal method in system modeling is the gap between text-based requirement and the initial specification. Thus, before modeling, a preliminary study of requirement analysis should be considered. There are several requirements engineering approaches used to describe requirements such as KAOs [11] and i* [13], but most of them stop at the requirements analysis stage, and do not involve later development process. Besides, SysML [2] is a modeling language for system engineering. Except for the basic diagram of UML, SysML also inherits the extensibility mechanism and provides some new diagrams, such as the requirement diagram and the parameter diagram. In [3], the author proposed to extend the SysML requirement diagram with goal model in KAOs method, and established a mapping relationships between the extended requirement diagram and the B method [12]. Considering that the B method is mainly applicable to software modeling, and the author didn't mention the consistency of the model. This paper proposes to combine SysML requirement diagram with Event-B modeling process, the paper mainly focuses on the contains relationship of SysML requirement diagram. First we use the requirement diagram to build the hierarchical relationships of requirements, then we translate these hierarchical relationships into the modeling process of Event-B, and verify the consistency of the model.

The organization of this paper is as follows, Sect. 2 gives some related knowledge of Event-B and SysML requirement diagram. Section 3 describes the methods that translates typical relationship of SysML requirement diagram into Event-B framework. Section 4 is a case study to illustrate the proposed approach. And Sect. 5 gives conclusion and future work.

2 Background

2.1 SysML

SysML [2] is a unified modeling language for complex system analysis and specification. As an extension of UML subset, some new diagrams are proposed such as requirement diagram and parametric diagram. In this paper, we focus on the requirement diagram. A requirement is represented by a specific identifier and a text-based description in requirement diagram. There are two kinds of relationships in requirement diagram, the relations *verify*, *satisfy*, and *refine* describe the relationship between requirements and other model elements [3]. The relation *contains* represents that a sub-requirement is a part of its composite requirement. The relation *derive* relate a derived requirement to its source requirement, for example a sub-system requirement may derived from a system requirement. The relation *copy* expresses that one requirement is the same version of another requirement. And these relations can be depicted as follows (Fig. 1):

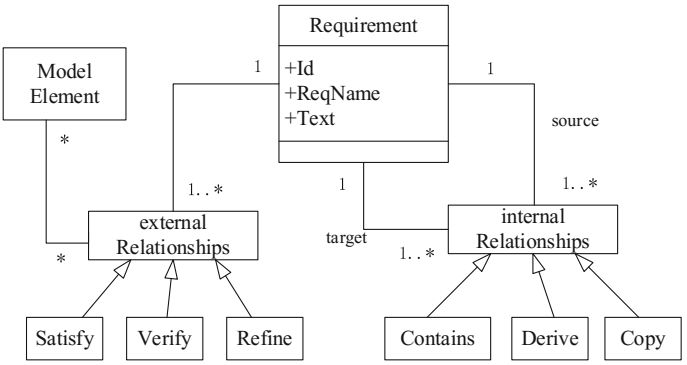

Fig. 1. Relations in SysML requirement diagram

2.2 Event-B

Event-B is a formal method for discrete system development based on first-order logic and set theory, which is evolved from the classical B method [1]. There are two main components in Event-B: context and machine. Context describes the static properties of the model. Machine describes the dynamic behavior of a model. Another important concept is the refinement mechanism in Event-B. A refinement process means more detailed properties introduced into the concrete model from the abstract model. The elements and the relationships of Event-B model can be shown as Fig. 2.

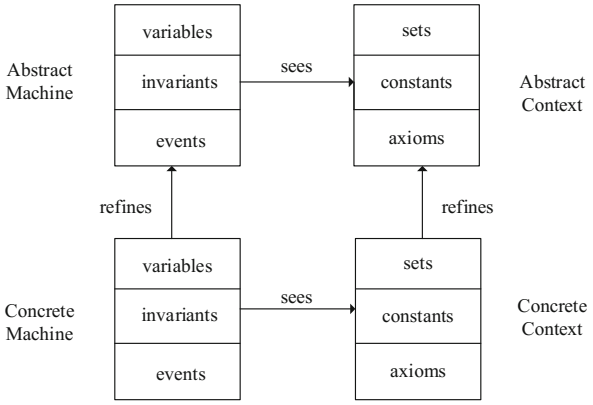

Fig. 2. Relations in Event-B model

In general, Event-B is a state-based discrete system modeling language, the mathematic model defined in machine is represented by states and its transition mechanism, i.e. the event. The state is represented by the value of the variable, and the property that states should always hold in a machine is invariant. In Event-B machines, events are used to update the state, the main elements of event are guards and actions.

Guards are conditions that the transition should satisfy, and actions are behaviors that update current states. A common form of an event is:

$$e \triangleq \text{any } x \textit{ where } G(s,c,x,v) \textit{ then } BA(s,c,x,v,v') \tag{1}$$

x is the parameter of the event, $G(s,c,x,v)$ is a set of conditions for triggering events, s is the carrier set and c is the constant, and v is the current value of the variable, respectively. The body of event e is $BA(s,c,x,v,v')$, where v' represents the updated value of the current state.

Through the refinement mechanism of Event-B, the abstract machine can be refined into a more specific machine. To maintain the consistency of the refinement chain in the model, a set of proof obligations (POs) should be proved, which is generated from the specification. There are two types of consistency in Event-B model, the model's self-consistency and the consistency with its abstract machine. If all POs are discharged, then the consistency of the model is confirmed.

After the model is built, it is necessary to prove that all the properties have been correctly added into the model. However, in a large project, the number of proofs may be up to thousands. Obviously, it is not possible to solve these proof manually. Rodin [4] is a development platform for Event-B, and it is based on eclipse. In rodin platform, many plug-ins are included for development, such as POs generator and prover, the first one is used to analyze the model and automatically generate corresponding proof obligations. The other is used to prove them.

3 From SysML Requirement Diagram to Event-B

Since the requirement in SysML requirement is textual and informal, it is not possible to directly translate requirement from requirement phase to formal specification phase. We propose to define rules to derive a refinement framework from the requirement diagram.

The main idea is to decompose the initial requirement into two different types, the functional requirement and the domain requirement. These two kind of requirement can be specified by contains relationship. The functional requirement is used to specify the intended behaviors that system will achieve, the domain requirement specifies the static environment factors such as the physical law the system should obey. As we have mentioned before, Context describes the static property, while machine describes the dynamic behavior of the system. Context can be extended with more properties while a domain requirement in SysML requirement diagram can be decomposed into more detailed sub-requirement [8]. The static property in domain requirement can be described by the axiom in context. In Event-B, the dynamic behavior is expressed by events and the invariants, in which the invariant is to make sure that the state converted by event is consistent with the model. The basic mapping relation from SysML requirement diagram to Event-B models is illustrated in Fig. 3.

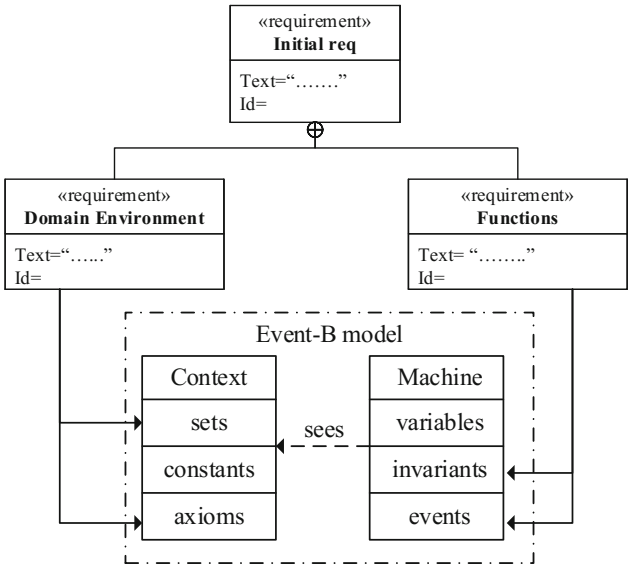

Fig. 3. Mapping relations between SysML requirement diagram and Event-B

3.1 Event-B Machine Consistency

In order to verify whether the model satisfies specified properties, Event-B defines a set of proof obligations that need to be discharged. And some POs involve the consistency and deadlock-freeness of model, such as Feasibility (FIS), invariant preservation (INV), deadlock-freeness (DLF). FIS is used to make sure that actions in events are feasible, INV is used to ensure that each event in machine maintains the property preserved in invariants, and DLF is to ensure that there are always some enabled events during the execution. The formal forms of these proof obligations are shown as follow:

$$\text{FIS: } A(s,c) \wedge I(s,c,v) \wedge G(s,c,v,x) \vdash \exists v' \cdot BA(s,c,v,x,v') \tag{2}$$

$$\text{INV: } A(s,c) \wedge I(s,c,v) \wedge G(s,c,v,x) \wedge BA(s,c,v,x,v') \vdash inv(s,c,v') \tag{3}$$

$$\text{DLF: } I(s,c,v) \vdash \bigvee_{i=1}^{n} (\exists x_i \cdot G(x_i,v)) \tag{4}$$

$A(s,c)$ is the axiom in context, $I(s,c,v)$ is invariants in machine, and $inv(s,c,v')$ are invariants that involve variables in $BA(s,c,v,x,v')$. According to the proofs in [14], the consistency property can be verified by discharging FIS and INV proof obligations. Meanwhile, with the help of plug-ins in rodin platform, these proof obligations can be automatically generated and discharged.

4 Case Study: A Leader Election Algorithm

The object of this section is to illustrate the approach through a common leader election algorithm from [1]. As we know that the leader node is a coordinator of a bunch of servers, there should be only one leader node and all servers should recognize the leader. In a word, the leader election is used to elect a leader node in a group of process.

In this paper, we consider to model a simple distributed system, the ring network. Each process in this ring network have their own id, and is able to send a message to the next process in this ring network, in addition, all processes can store the message in their buffers. In the algorithm, the process only accepts the message which is no less than its own id and rejects messages that have smaller id. The algorithm stops when a process receives its own id from other node, and this node is the leader node.

4.1 The Initial Model

At the beginning, we have an initial requirement described as "A leader node should be elected in a ring network". From this initial requirement, we can derive two sub requirements, shown as Fig. 4.

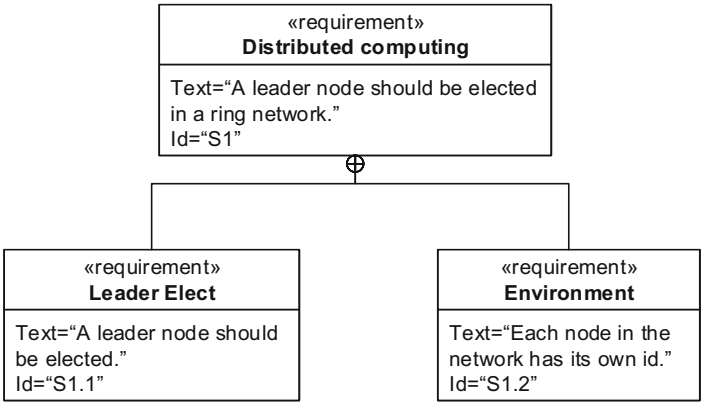

Fig. 4. Initial requirement diagram

The sub-requirement "Leader Elect" and "Environment" can be mapped into Event-B machine and context, respectively. Here, we consider to build a simple network which contains a set of nodes, and the corresponding specification in Event-B context is shown as Fig. 5.

In this context, we defined a constant *Num* to include nodes with different id. The axm4 means that there should always be a node that have the biggest id. Moreover, we don't consider about the ring structure. It will be refined into next refinement.

And the next requirement is "Leader Elect", in this event, *maxId* is a variable that can be assigned as the largest id in all nodes. The basic form can be shown as follows (Fig. 6):

$constants\ Num$
$axioms$
@axm1 $Num \subseteq \mathbb{N}$
@axm2 $finite(Num)$
@axm3 $Num \neq \phi$
@axm4 $\exists a \cdot \forall x \cdot x \in Num \Rightarrow a \geq x$

Fig. 5. Initial context in Event-B

event elect
then
@act1 $maxId := max(Num)$
end

Fig. 6. Event elect

4.2 First Refinement

From the initial model, we have built a simple model that contains only one event and don't consider the ring structure. The requirement diagram should be extended further, as the requirement can be explained in a detailed way. In the following refinement a ring structure should be added into the context, and the requirement environment will be extended as follows (Fig. 7):

And the corresponding context in Event-B model is shown as Fig. 8.

The constant *next* is a function that maps one node to another node.

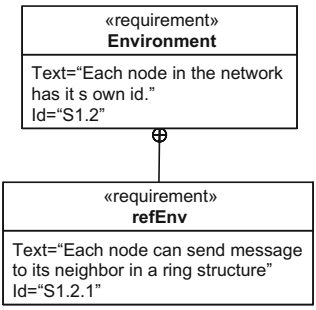

Fig. 7. Extend the Env requirement

axm1_1: $next \in Num \twoheadrightarrow Num$

axm1_2: $\forall S \cdot next^{-1}[S] \subseteq S \wedge S \neq \phi \Rightarrow Num \subseteq S$

Fig. 8. Extended context in Event-B

Also, the requirement *Leader Elect* will be refined into three more concrete requirements, *accept, reject* and *refElect*, which can be depicted as follows (Fig. 9):

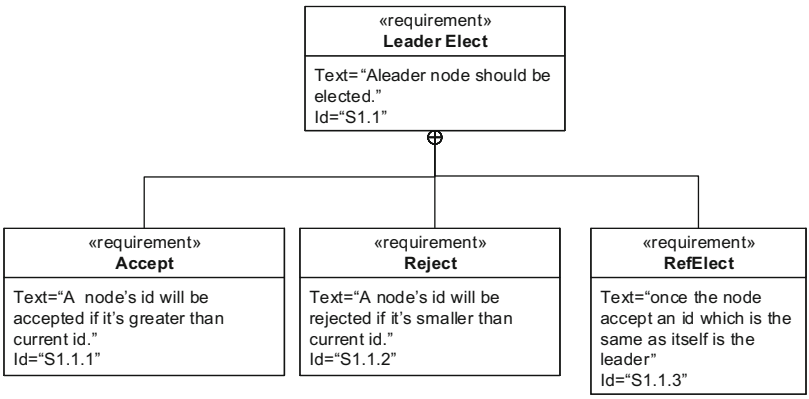

Fig. 9. Extended Leader Elect

In this requirement diagram, three refined requirements are contained in the *LeaderElect*. The requirement *Accept* means that nodes in the ring network only receive message that has bigger id. On the contrary, the requirement *Reject* rejects message that is smaller than its own id. And the requirement *RefElect* details the operation in *LeaderElect*. The corresponding events in the refined machine are described partially as follows (Figs. 10 and 11):

event Accept
any x
where
@grd1_1 $x \in dom(buffer)$
@grd1_2 $buffer(x) < x$
then
@act1_1 $buffer(x) := next(buffer(x))$
end

Fig. 10. Event Accept

Event RefElect
any x
where
@grd1_1 $x \in dom(buffer)$
@grd1_2 $x = buffer(x)$
then
@act1_1 $maxId \coloneqq x$
end

Fig. 11. Event RefElect

In these events, variable *buffer* is used to store the nodes that are not rejected, and the invariant is shown as follows (Fig. 12):

Inv1_1: $buffer \in Num \nrightarrow Num$
Inv1_2: $\forall x \cdot x \in dom(buffer) \Rightarrow x = max(i\left(x \mapsto next^{-1}\left(buffer(x)\right)\right))$

Fig. 12. Invariants in refined machine

With this refinement step, a concrete model which contains more detailed information can be built. The next step would be considering the consistency of this model.

4.3 Model Verification

Although the ring network model has been built through refinement, we still have to verify the consistency and correctness of our model. In Event-B modeling, there are set of proof obligations that should be proved while the model has been build. For example, after the Accept event has been executed, we have to proof that the new value of variable a is still consistent with the corresponding invariant such as inv1_1. And the proof process can be shown as follows (Fig. 13):

Inv1_1
Guards of event
Accept
⊢
Modified invariant Inv1_1

$buffer \in N \nrightarrow N$
$x \in dom(buffer)$
$buffer(x) < x$
⊢
$(\{x\}\ buffer) \cup \{x \mapsto n\left(buffer(x)\right)\}$
$\in N \nrightarrow N$

Fig. 13. Event Accept INV1_1 proof obligation

If all the proof should be proved manually, it would be a long time and becomes difficult. Rodin [4] is a platform for Event-B modeling, it provides not only a development environment, but also some tools that can automatically prove POs of the model, which simplified the proof procedure [5, 6]. The ring network we have built is automatically proved by Rodin platform, which can be shown as follows (Fig. 14):

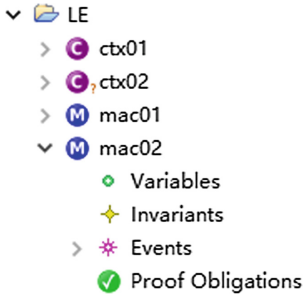

Fig. 14. All POs discharged

5 Conclusion and Future Work

As a formal specification language, Event-B is capable for complex system modeling. However, one of the major problem is that there is no standard guideline to use the refinement mechanism. On the other hand, SysML requirement diagram gives a preliminary analysis of requirements, and the hierarchical structure of requirements is built. In this paper, we propose mapping relationships between SysML requirement diagrams and Event-B refinement process. And since these relationships are partial, a more precise semantic should be added in SysML requirement relations. Here, we plan to extended SysML requirement diagram with goal model in KAOs method. Goals in goal model can be specified into LTL formula, as LTL can describe both safety property and liveness property, we will give a more precise preliminary analysis of requirements. Our future work will mainly concentrate on the translation of extended requirement diagram and Event-B model elements.

Acknowledgement. This work was supported by the National High-tech R&D Program of China (863 Program) under Grant No. 2015AA015303; Project 61272083 supported by National Natural Science Foundation of China; Supported by National key research and development program 2016YFB1000802.

References

1. Abrial, J.R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press, Cambridge (2010)
2. Friedenthal, S., Moore, A., Steiner, R.: A practical guide to sysml. San Francisco Jung Inst. Lib. J. **17**(1), 41–46 (2012)

3. Laleau, R., Semmak, F., Matoussi, A., Petit, D., Hammad, A., Tatibouet, B.: A first attempt to combine sysml requirements diagrams and b. Innov. Syst. Softw. Eng. **6**(1–2), 47–54 (2010)
4. Butler, M., Hallerstede, S.: The Rodin formal modelling tool. In: The International Conference on Formal Methods in Industry, p. 2. British Computer Society (2007)
5. Le, H.A., Thi, L.D., Truong, N.T.: Modeling and verifying imprecise requirements of systems using Event-B. In: Huynh, V., Denoeux, T., Tran, D., Le, A., Pham, S. (eds.) Knowledge and Systems Engineering. Advances in Intelligent Systems and Computing, vol. 244, pp. 313–325. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-02741-8_27
6. Younes, A.B., Hlaoui, Y.B., Ayed, L.J.B.: A meta-model transformation from UML activity diagrams to Event-B models. In: IEEE International Computer Software and Applications Conference Workshops, pp. 740–745. IEEE Computer Society (2014)
7. Bousse, E., Katsuragi, T.: Aligning SysML with the B method to provide V&V for systems engineering. In: The Workshop on Model-Driven Engineering, Verification and Validation, pp. 11–16. ACM (2012)
8. Mentré, D.: SysML2B: automatic tool for B Project Graphical Architecture Design Using SysML. In: Butler, M., Schewe, K.-D., Mashkoor, A., Biro, M. (eds.) ABZ 2016. LNCS, vol. 9675, pp. 308–311. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-33600-8_26
9. Xu, H.: Model based system consistency checking using Event-B. Comput. Softw. (2012)
10. Krakora, J., Waszniowski, L., Pisa, P., Hanzalek, Z.: Timed automata approach to real time distributed system verification. In: IEEE International Workshop on Factory Communication Systems, Proceedings, pp. 407–410. IEEE (2004)
11. Lamsweerde, A.V.: Requirements engineering: from system goals to UML models to software specifications. Wiley Publishing, Hoboken (2009)
12. Abrial, J.R.: The B-Book - Assigning Programs to Meanings. DBLP (1996)
13. Yu, E.S.K., Mylopoulos, J.: Understanding "why" in software process modelling, analysis, and design. In: International Conference on Software Engineering, pp. 159–168. IEEE Computer Society Press (1994)
14. Traichaiyaporn, K., Aoki, T.: Preserving correctness of requirements evolution through refinement in Event-B. In: Software Engineering Conference, Vol. 1, pp. 315–322. IEEE (2014)