2021 International Symposium on Theoretical Aspects of Software Engineering (TASE) | 978-1-6654-4163-6/21/$31.00 ©2021 IEEE | DOI: 10.1109/TASE52547.2021.00034

RE2B: Enhancing Correctness of Both Requirements and Design Models

Shiling Feng, Xiaohong Chen, Qin Li* and Yongxin Zhao
Shanghai Key Laboratory of Trustworthy Computing
East China Normal University
Shanghai, China

***Abstract*—Software requirements are the criteria for the correctness of the software behaviors. How to verify the correctness of the requirements is a big challenge in requirements engineering. Among various requirements approaches, the environment modeling based requirements engineering(EBRE) gains great popularity due to its explicit environment model. However, it still lacks of a formal approach to verify the correctness of the requirements models proposed in EBRE. Event-B is a popular formal method employing step-wise refinement to construct system models and verify their correctness according to the system requirements. This paper combines EBRE with Event-B to merge the advantages of the two methods. On the one hand, we transform the EBRE models to the initial Event-B model to guide the further design and development of the system. On the other hand, the transformed Event-B model can provide formal proof on the consistency of the requirements model. We also implement a plug-in tool on the Rodin platform to realize the automatic transformation from EBRE models to Event-B model and to commit a formal verification on the correctness of the transformed requirements model.**

***Index Terms*—Event-B, Environment Modeling based Requirements Engineering, Requirements Verification**

I. Introduction

Requirements are the key criteria for the correctness of a software system. Requirements Engineering (RE) is a critical process of software engineering before software design and development. Detailed and accurate requirements analysis will help software designers better understand the requirements and avoid errors from early stage. However, how to verify the correctness of the RE models remains a big challenge.

The mainstream approaches of RE include goal oriented approaches [1], agent and intention oriented approaches [2] [3], scenario based approaches [4] and the Problem Frames(PF) approach [5]. Among these approaches, PF approach is famous for its environment perspective, and is particularly suitable for environment interactive sensitive systems like Cyber-Physical systems [6]. PF emphasizes the necessity to characterize the real world in which the software system will function and refers the meaning of the requirements to the description of the relevant problem domains of the real world. Based on PF, Jin further proposes the environment modeling based RE(EBRE) [7] [8]. EBRE explicitly model interactive environment, and provides systematic process and RE models for requirements elicitation and analysis based on the environment model. But it lacks of formal verification to guarantee the correctness of the requirements models.

Event-B [9] is a famed formal method for software system design and verification. It can automatically generate proof obligations(POs) for verifying the correctness of the model construction, which can significantly improve the quality of software systems. There has been a lot of works devoted to combining requirements analysis and graphical semantics with Event-B [10] [11]. For example, combing PF with Event-B [12], and combing UML with Event-B [13]. They only verify the correctness of requirements model, such as use cases and problem diagram. However, they do not provide support to verify restrictions coming from the environment because they do not have an environment model. This is urgently required in the environment sensitive systems such as CPS.

In this paper, we combine EBRE with Event-B to merge the advantages of the two methods. We propose a set of transformation rules to automatically transform requirements models defined by EBRE to Event-B models. It complements the Event-B approach by providing an initial design model obtained directly from requirements. The verification mechanism of Event-B is employed to verify the correctness of requirements model in EBRE. A plug-in tool for Rodin platform is developed to support the transformation and verification process. The approach can enhance the correctness of both requirements and design models, which can greatly improve the quality of software development.

The main contributions of the paper are as follows:

- A set of transformation rules to transform EBRE requirements models to Event-B models are defined.
- We investigate the correspondence between the POs and the properties of requirements and employ the verification technique of Event-B to verify the correctness of the requirements model.
- A plug-in tool for Rodin platform is developed to support the automatic transformation from EBRE to Event-B and commit verification for the correctness of requirements.

The rest of the paper is organized as follows. Section II briefly introduces the core concepts of EBRE and Event-B. Section III provides the specific semantic transformation rules

* Qin Li is the corresponding author (qli@sei.ecnu.edu.cn). This work is supported by National Key Research and Development Program (2020AAA0107800).

978-1-6654-4163-6/21/$31.00 ©2021 IEEE
DOI 10.1109/TASE52547.2021.00034

from EBRE model to Event-B model. Section IV introduces the verification technique for verify the correctness of the requirements. The transformation rules are demonstrated on a running example of light unit control system. Section V outlines the related work. Section VI concludes the paper and mentions some future work.

II. Background

EBRE provides a set of models to support requirements elicitation, problem projection and specification generation which could be found in [14]. The models includes *Environment Ontology*, *Problem Diagram*, and *Scenario Diagram*. The Environment Ontology provides the basic concepts used in the problem description. The environment is composed by a set of environment entities, which will interact with the software system. It extracts concepts that describe the environment entities, such as *attribute*, *state*, *state transition*, etc., as well as their relations. Fig.1 describes the domain environment ontology of light unit control.

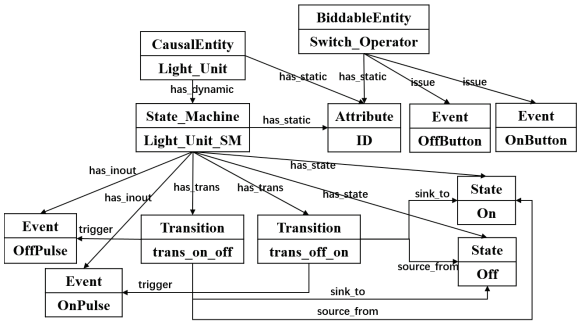

Fig. 1. Domain Environment Ontology of light unit control.

In addition to showing the software systems, problem domains, and behavior interactions of the software problem to be developed, the Problem Diagram further shows the requirements and requirements references of the software problem to be developed. The task of realization a requirement is to construct a scenario that describe how the software system interacts to make expected changes in the problem domain. There is a simple example about light control depicted in the Scenario Diagram as shown in Fig.2. A Scenario Diagram contains different kinds of nodes and line relationships, which manifest as dependencies between phenomena. Solid line nodes represent a actual behavior interactions set (Beh). The line that connects between two solid line nodes specifies behavior order ($behOrd : Beh \rightarrow Beh$). Dotted line nodes represent an expected interaction set (Exp). Dotted line connections between dotted line nodes represent the expected order ($expOrd : Exp \rightarrow Exp$). The expected interaction can also represents a state change. Besides, a scenario diagram also contains edges linking solid line nodes and dotted line nodes. The synchronous relation ($synchrocity : Beh \leftrightarrow Exp$) states that the two nodes connected represent the synchronous interaction, such as the relation between $int1$ and $int1$. The behavior enable relation ($behEna : Beh \rightarrow Exp$) describes that behavior interaction enables expected interaction, such as the relation between $int3$ and $int5$, which is also the property described by the state machine in the environment ontology.

On the other hand, the scenario diagram also contains some composite nodes, such as the *Merge* node, the *Branch* node. The *Decision* node. The *Merge* node specifies the structure where different phenomena can trigger the same phenomenon. The *Branch* node specifies the structure where the sharing phenomenon can trigger different branching phenomena. The *Decision* node is also represents a branch structure except that the trigger of different branching phenomena depends on whether the decision condition is satisfied or not.

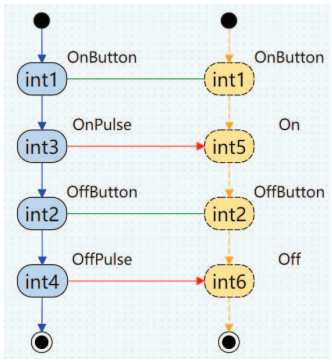

Fig. 2. Example of Scenario Diagram.

Fig.3 shows a common structure of Event-B models. It consists of two main components: *Context* and *Machine*. The Context defines constants, which can be values or dataset. The Machine mainly depicts dynamic behaviors, including states, state attributes, and events. Refinement is to increase the functionality of the system, add details, and change the state model. As you refine a Machine, the new variables and new events might be added. Generally, a model requires multiple refinements to meet the requirements.

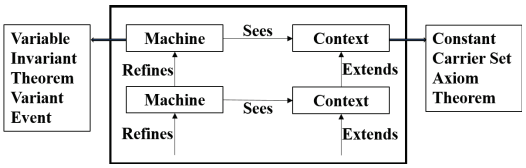

Fig. 3. Model structure of Event-B.

III. Transformation from EBRE to Event-B

We propose a framework shown in Fig.4. to integrate the EBRE and Event-B method. The environment ontology (EO) and scenario diagrams (SD) in EBRE model are transformed to corresponding components in Event-B model through a number of transformation rules. The formal verification committed on Event-B model provides corresponding feedbacks to the requirement model regarding its correctness. The proposed transformation rules are defined in detail below.

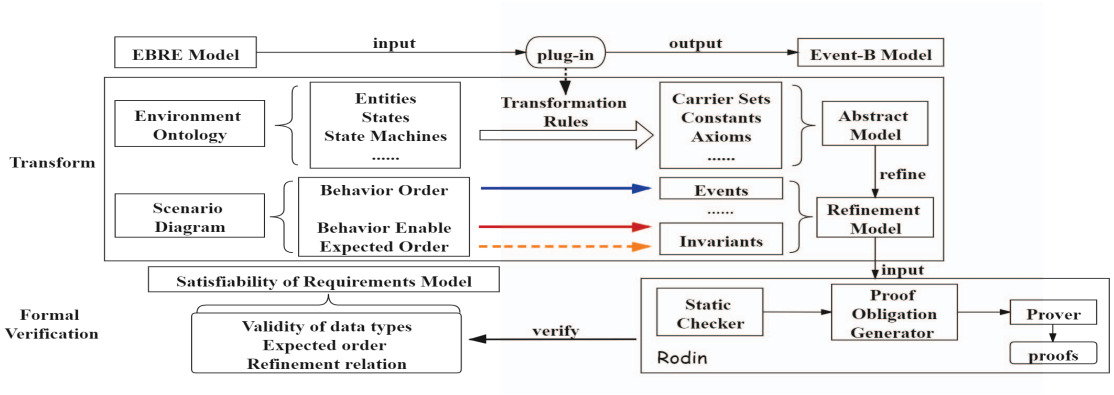

Fig. 4. An overview of RE2B framework.

A. Environment Ontology

1) Basic Treatment: The static components in the EO such as the states, environment entities and their associations are transformed to corresponding components in the **context** models of Event-B. The dynamic components in the EO such as attributes of entities and state transitions are transformed to the corresponding components in the **machine** models of Event-B. Fig.5 lists the indexes of the transformation rules that transform EO to Event-B model.

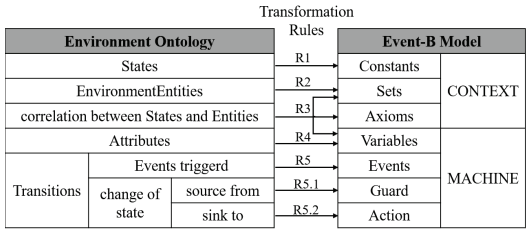

Fig. 5. Transformation rules from EO to Event-B.

Rule **R1** transforms each value of states defined in EO to a constant in Event-B context.

Rule **R2** transforms every environment entity in EO to a carrier set in Event-B context.

In EO model, an environment entity is associated with a state machine specifying the entity states and the transitions that change these states. Rule **R3** introduces an axiom in Event-B context model to capture the *has_state* association between the state and the state machine. The transformation scheme of **R1-3** is described in Fig.6 where the blue words are reserve words of Event-B and the green words are extracted from EO by the transformation rules.

The attributes of entities, such as ID are also associated with the entity. We map attributes to variables in the machine model of Event-B using rule **R4**.

Rule **R5** transforms the state transitions in EO state machine to events in Event-B model. Concretely, the state that *source from* can be mapped to a guard of the event (i.e., **R5.1**), and the state that *sink to* can be mapped to an action of the event

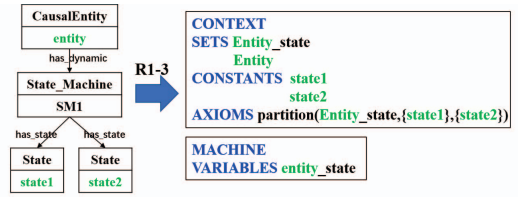

Fig. 6. Transformation scheme of R1-3.

(i.e., **R5.2**). The transformation scheme of **R5** is shown in Fig.7.

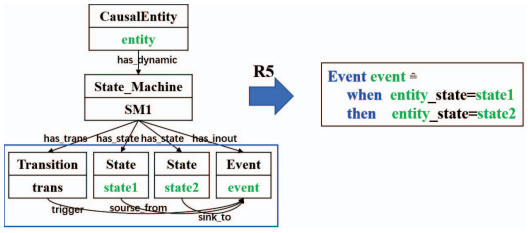

Fig. 7. Transformation scheme of R5.

The Event-B models we transformed from EO capture the constraints to the software behaviors from the view of environment. The system design and development should subject to these constraints. Therefore, we take these models as the abstract models in Event-B, and further refine them by introducing more constraints from the scenario diagrams of EBRE. We transform EO of light control in Fig.1 according to the above rules, as shown in Fig.8.

B. Scenario Diagram

1) Basic Treatment: The SD model mainly specifies the execution order of phenomena in a scenario and the correspondence relationship between the actual behaviors and the expected phenomena. Based on the abstract models transformed from EO, we extract the information stated in the SD to obtain a refined **machine** in Event-B. Fig.9 lists the index of the transformation rules that transform basic structures in SD to Event-B model. Rule **R6** transforms each phenomenon in the

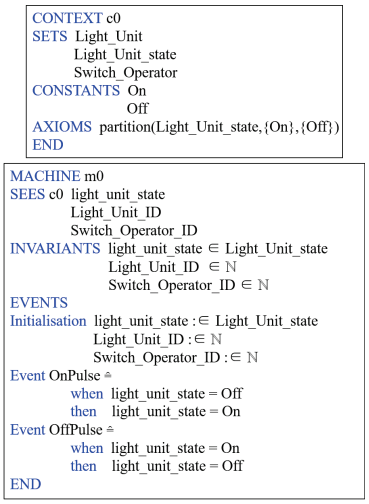

Fig. 8. Transformation from EO of Fig.1 to Event-B.

behavior order of the SD to an *event* in Event-B machine. Special conditions are introduced to the *guard* and effect of the *event* to capture the specified behavior order. Rule **R7** transforms the expected phenomena and their order relation to *invariants* of the Event-B model. It enables the verification on the consistency between behavior order and expected order. The detail of rule **R6** and **R7** are shown in next subsection.

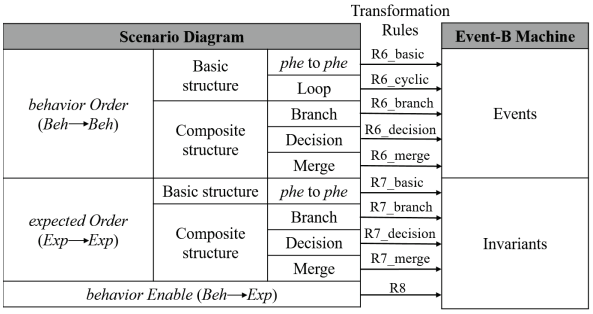

Fig. 9. Transformation rules from SD to Event-B.

2) Basic structure: The basic structure in SD explicitly specify the sequential execution order of phenomena. For example, the structure $\boxed{phe1} \rightarrow \boxed{phe2}$ means that $phe2$ can be triggered after the execution of $phe1$. However, it is not trivial to transform such order to Event-B model directly since Event-B model does not have similar notations for specifying execution order of events. Instead, the execution of events in Event-B relies on the satisfaction of its guard condition in current state.

To tackle this problem, we employ a mechanism to restrict the execution order of the events in Event-B. We introduce an instrumental variable *count* for each event and record the times that the event has been triggered. For example, to realize the sequential execution order $\boxed{phe1} \rightarrow \boxed{phe2}$ for any event $phe1$ and $phe2$, we introduce instrumental variables $phe1\_count$ and $phe2\_count$ to record the trigger times of $phe1$ and $phe2$ respectively. These count variables are set to 0 in initialization and increased by 1 in the action of their events. To specify the order between the two event, we add a guard condition $phe1\_count = phe2\_count + 1$ in $phe2$ to guarantee that $phe2$ can be triggered after $phe1$. We add a guard condition $phe1\_count = phe2\_count$ in $phe1$ to guarantee that $phe1$ can not be triggered again until $phe2$ is triggered.

R6_basic: For directly connected phenomena in behavior order, the latter phenomenon can be triggered after the former phenomenon triggered, and the former can be triggered again only when the latter is triggered. The specific transformation scheme is shown in Fig.10.

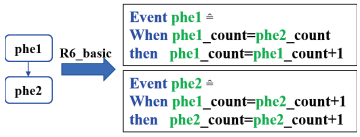

Fig. 10. R6_basic from SD to Event-B.

R7_basic: For directly connected phenomena in expected order, the trigger times of the two phenomena are equal or differ by 1, and the source phenomenon must be triggered earlier. Its transformation scheme is shown in Fig.11.

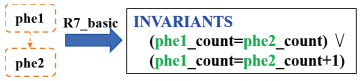

Fig. 11. R7_basic from SD to Event-B.

R6_cyclic: There is a start phenomenon and an end phenomenon in behavior order of SD. The control flow starts from the start phenomenon and when it triggers end phenomenon, it returns to the start phenomenon to form a cycle. On the other hand, it requires that the start phenomenon can only be triggered again after the end phenomenon. In order to cover this property, we make a special treatment to the *guard* condition for the event corresponding to the start phenomenon. The transformation scheme is described in Fig.12.

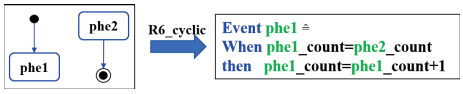

Fig. 12. R6_cyclic from SD to Event-B.

Rule **R8** transforms the behavior enable structure in SD to an *invariant* in Event-B model. It reflects the synchronous property between the triggering of an event and the change of an entity state. The transformation scheme of **R8** is described in Fig.13.

The SD in Fig.2 describes a scenario for artificially controlling the light unit state. Once we turn on the button, the state of the light unit can be changed into On. When the light is on, we can correspondingly turn off the button to change the

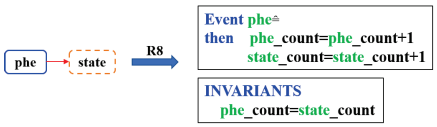

Fig. 13. R8 from SD to Event-B.

light unit into Off. According to the above transformation schemes for the basic structure, we transform the SD model to Event-B model as shown in Fig.14.

```
INVARIANTS
   (OnButton_count=On_count)
                ⩔ (OnButton_count=On_count+1)
   (On_count=OffButton_count)
                ⩔ (On_count=OffButton_count+1)
   (OffButton_count=Off_count)
                ⩔ (OffButton_count=Off_count+1)

Event OnButton ≙
        when  OffPulse_count=OnButton_count
              OnButton_count=OnPulse_count
        then  OnButton_count=OnButton_count+1
Event OnPulse ≙ Refines OnPulse
        when  light_unit_state = Off
              OnButton_count=OnPulse_count+1
              OnPulse_count=OffButton_count
        then  light_unit_state = On
              OnPulse_count=OnPulse_count+1
              On_count=On_count+1
Event OffButton ≙
        when  OnPulse_count=OffButton_count+1
              OffButton_count=OffPulse_count
        then  OffButton_count=OffButton_count+1
Event OffPulse ≙ Refines OffPulse
        when  light_unit_state = On
              OffButton_count=OffPulse_count+1
        then  light_unit_state = Off
              OffPulse_count=OffPulse_count+1
              Off_count=Off_count+1
```

Fig. 14. The Event-B model tranformed from SD in Fig.2.

3) Composite structure: We introduce different transformation rules for three different composite structures: branch structure, decision structure and merge structure. These composition structure can appear in both actual behavior order and expected behavior order of SD.

R6_branch: The branch structure in behavior order (shown in Fig.15) connects a sharing phenomenon phe and two or more branching phenomena br_1 to br_n. All of these branching phenomena can be triggered when the sharing phenomenon is triggered. We capture this structure with the manipulation of the event guard condition using count variables. The transformation scheme for every branching event is shown in Fig.15.

R7_branch: The branch structure in expected order is transformed to an invariant saying that the trigger times of sharing phenomenon is equal or one more than the sum of all branching phenomena. The transformation scheme is described in Fig.16.

R6_decision: The decision structure shown in Fig.17 connects a sharing phenomenon and two branching phenomenon with a decision condition. If the decision condition is satisfied after phe, the left branch is triggered. Otherwise, the right branch is triggered. The transformation scheme for every branching phenomenon is described in Fig.17.

R7_decision: For the decision structure in expected order (shown in Fig.18), we need to add corresponding *invariants* specifying (1) br_1, br_2 is triggered after phe; (2) if br_1/br_2 is triggered, the decision condition must be $true/false$ in that state. The transformation scheme is shown in Fig.18.

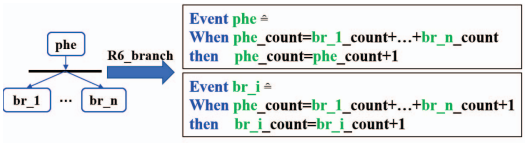

Fig. 15. R6_branch from SD to Event-B.

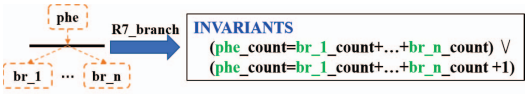

Fig. 16. R7_branch from SD to Event-B.

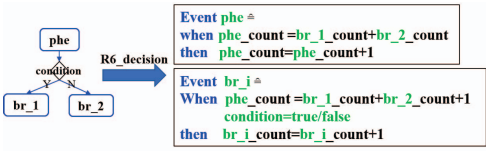

Fig. 17. R6_decision from SD to Event-B.

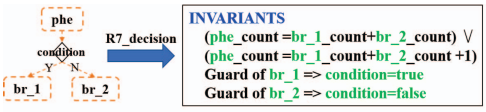

Fig. 18. R7_decision from SD to Event-B.

R6_merge: The merge structure in behavior order (shown in Fig.19) connects two or more branching phenomena mr_1 to mr_n and a sharing phenomenon. The sharing phenomenon can be triggered when any of the branching phenomena is triggered. We capture this structure with the event guard condition (shown in **R6_merge0**). Besides, in order to clearly indicate the specific branch phenomenon that trigger the sharing phenomenon, we decompose the sharing phenomenon into phenomena phe_1 to phe_n. phe_i is triggered after mr_i. The specific transformation scheme for the sharing phenomenon is shown in Fig.19.

R7_merge: The merge structure in expected order is transformed to invariants saying that the sum trigger times of all branching phenomena is equal or one more than that of the sharing phenomenon (shown in **R7_merge0**). Besides, we need to add corresponding invariants specifying (1) phe_i is triggered after mr_i; (2) the sum trigger times of each refined phenomenon phe_i is equal to that of phe. The transformation scheme is shown in Fig.20.

Fig.21 is the SD of light unit control, which describes two ways to control the light unit state. Both the artificially turning on the button and the reception of the $7PM$ signal can trigger the pulse on, and the light can be turned On when the pulse is on. When the light is On, the pulse can be turned off when the

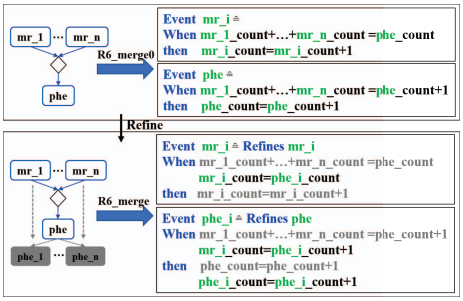

Fig. 19. R6_merge from SD to Event-B.

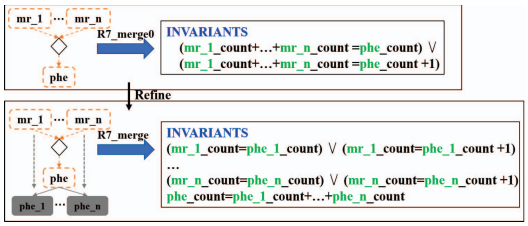

Fig. 20. R7_merge from SD to Event-B.

button is artificially turned off or the $9PM$ signal received, the state of the light will directly be changed into Off. We perform the transformation according to the well-defined transformation rules described above as shown in Fig.22-23. $OffPulse$ is refined in the same way as $OnPulse$.

There may be multiple composite nodes associated with phenomena. We set up virtual nodes that are different from the way general phenomena named, and then convert into the basic treatment for modeling. As shown in Fig.24(a), when creating *events* for $phe2$ and $phe5$, the guard condition for phenomena becomes complicated when the *Merge* node and *Decision* node appear in combination. We add virtual nodes $phe01$ and $phe02$, as shown in Fig.24(b). The sub-scenario diagram is divided into three basic structure, then the modeling of $phe2$ and $phe5$ can be transformed according to the previously defined rules.

The transformation from EBRE to Event-B assigns a formal

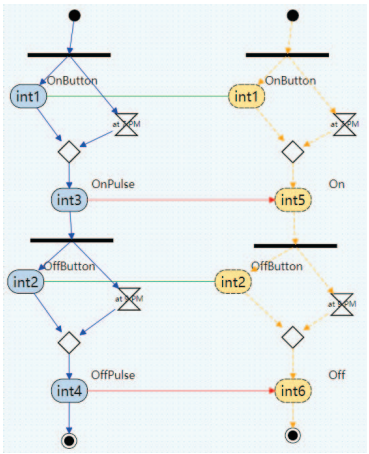

Fig. 21. SD of light unit control.

```
INVARIANTS
(at_7PM_count+OnButton_count=On_count)
        ∨ (at_7PM_count+OnButton_count=On_count+1)
(On_count=at_9PM_count+OffButton_count)
        ∨ (On_count=at_9PM_count+OffButton_count+1)
(at_9PM_count+OffButton_count=Off_count)
        ∨ (at_9PM_count+OffButton_count=Off_count+1)
OnPulse_count=On_count
OffPulse_count=Off_count

Event OnButton ≙
    when OffPulse_count=OnButton_count+at_7PM_count
         OnButton_count+at_7PM_count=OnPulse_count
    then OnButton_count=OnButton_count+1
Event at_7PM ≙
    when OffPulse_count=OnButton_count+at_7PM_count
         at_7PM_count+ OnButton_count=OnPulse_count
    then at_7PM_count= at_7PM_count+1
Event OnPulse ≙ Refines OnPulse
    when light_unit_state = Off
         OnButton_count+at_7PM_count=OnPulse_count+1
         OnPulse_count=OffButton_count+at_9PM_count
    then light_unit_state = On
         OnPulse_count=OnPulse_count+1
         On_count=On_count+1
Event OffButton ≙
    when OnPulse_count=OffButton_count+at_9PM_count+1
         OffButton_count+at_9PM_count=OffPulse_count
    then OffButton_count=OffButton_count+1
Event at_9PM ≙
    when OnPulse_count=OffButton_count+at_9PM_count+1
         at_9PM_count+OffButton_count=OffPulse_count
    then at_9PM_count= at_9PM_count+1
Event OffPulse ≙ Refines OffPulse
    when light_unit_state = On
         OffButton_count+at_9PM_count=OffPulse_count+1
    then light_unit_state = Off
         OffPulse_count=OffPulse_count+1
         Off_count=Off_count+1
```

Fig. 22. Abstract model for SD of Fig.21 in Event-B.

```
INVARIANTS
   (OnButton_count=On1_count) ∨ (OnButton_count=On1_count+1)
   (at_7PM_count=On2_count) ∨ (at_7PM_count=On2_count+1)
   OnPulse_count=OnPulse1_count+OnPulse2_count
   On_count=On1_count+On2_count
   OnPulse1_count=On1_count
   OnPulse2_count=On2_count

Event OnButton ≙ Refines OnButton
    when OffPulse_count=OnButton_count+at_7PM_count
         OnButton_count+at_7PM_count=OnPulse_count
         OnButton_count=OnPulse1_count
    then OnButton_count=OnButton_count+1
Event at_7PM ≙ Refines at_7PM
    when OffPulse_count=OnButton_count+at_7PM_count
         at_7PM_count+ OnButton_count=OnPulse_count
         at_7PM_count=OnPulse2_count
    then at_7PM_count= at_7PM_count+1
Event OnPulse1 ≙ Refines OnPulse
    when light_unit_state = Off
         OnButton_count+at_7PM_count=OnPulse_count+1
         OnButton_count=OnPulse1_count+1
    then light_unit_state = On
         OnPulse_count=OnPulse_count+1
         On_count=On_count+1
         OnPulse1_count=OnPulse1_count+1
         On1_count=On1_count+1
Event OnPulse2 ≙ Refines OnPulse
    when light_unit_state = Off
         OnButton_count+at_7PM_count=OnPulse_count+1
         at_7PM_count=OnPulse2_count+1
    then light_unit_state = On
         OnPulse_count=OnPulse_count+1
         On_count=On_count+1
         OnPulse2_count=OnPulse2_count+1
         On2_count=On2_count+1
```

Fig. 23. Refined model for SD of Fig.21 in Event-B.

semantics to the EBRE model. The correctness of the transformation rules is mainly supported by the correspondence between the semantics of the two modelisms. Furthermore, the transformation facilitates the formal verification on the correctness of the requirements model in EBRE. The consistency between the verification result from Event-B and the analysis result from EBRE can be considered as another evidence for the correctness of the transformation rules.

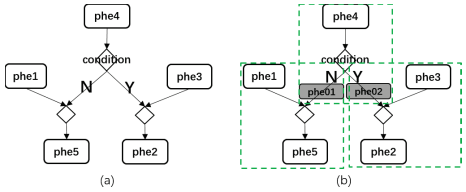

Fig. 24. Example of a sub-scenario diagram.

IV. Formal Verification

The correctness of the requirements model in EBRE relies on the validity within and the consistency between the environment constraints specified in EO and the system-environment interactions specified in SD. With the transformation rules proposed in Section III, we can automatically construct an Event-B model that covers all the constraints specified in EBRE model. Therefore, the verification technique provided by Event-B and its tool platform Rodin can be employed to prove the correctness of EBRE model if we can establish the generic correspondence between each constraint specified in EBRE and the proof obligation generated by Event-B.

Proof Obligations	Verified Properties for EMRE
INV for data and state variables	Validity of data and state types in EO
INV for guards and "count" variables	Consistency between actual behavior order and expected order in SD
GRD,SIM for new guards	Consistency between EO and SD

Fig. 25. Correspondence between POs and verified properties.

For environment ontology, the main properties we need to verify are the validity of the types of the entity states. Since the type information is encoded as the *invariants* on corresponding variable in the Event-B model, a series of POs will be generated by Event-B to check whether these invariants are preserved during the entire execution of the system.

For scenario diagrams, the key property we need to verify is whether the behavior order satisfy the expected one. According to Section III, the expected order is specified with invariants on the instrumental *count* variables while the actual behavior order is manipulated through the guard conditions of the Event-B events. Hence, proving the POs for preserving such invariants can guarantee the consistency.

Finally, the consistency between EO and SD can be verified by checking the refinement relationship between the models transformed from EO and SD. The POs relating to the state refinements and event refinements can be used to checking this consistency. Fig.25 shows the correspondence between the POs and the properties to be verified by the SD.

We can verify the satisfiability of the requirements model according to whether the POs generated by the *invariants* is proved to be correct. The verification statistics of the correct case for light control are shown in Fig.26. If the requirements model is not satisfied, the specific unsatisfied property in SD can be located according to the POs that failed to be proved.

Element Name	Total	Auto	Manual	Reviewed	Undischarged
lightUnit	**83**	**83**	**0**	**0**	**0**
c0	0	0	0	0	0
m0	3	3	0	0	0
m1	32	32	0	0	0
m2	48	48	0	0	0

Fig. 26. Verification result of light unit control in Event-B.

An example of the error SD of Fig.2 is shown in Fig.27(a). The proof results shown in Fig.27(b) are used to verify the consistency between actual behavior order and the expected one in (a). It can be seen that two POs failed to be proved. We locate the specific invariants *inv9* and *inv10* in Event-B. The specific description of *inv9* is $(On\_count = OnButton\_count) \vee (On\_count = OnButton\_count + 1)$, which satisfy the property that the state of the light turns On before the phenomenon $OnButton$ is triggered in expected order. However, in the actual behavior order, when phenomenon $OnButton$ is triggered, $OnPulse$ can be triggered, and $OnPulse$ directly causes the state of the light to be On. There's an inconsistency between the two order relations. The same is true for $inv10$. We have also completed the implementation of this error location function in the plug-in tool as shown in Fig.28.

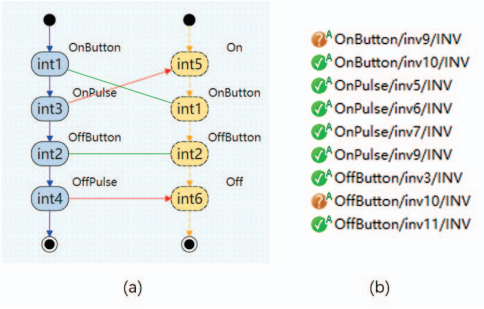

Fig. 27. Example of wrong SD and its proof results.

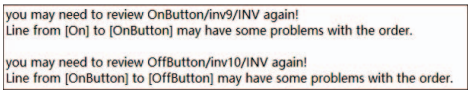

Fig. 28. Example of error location.

V. Related Work

A number of researches study the connection between Event-B models and requirement models to verify the correctness of software design. Butler et al. [10] considers an extension of Event-B with an event ordering mechanism. They rely on an Eclipse library to manipulate EMF models using graphical editors instead of CSP-like language notation. Additional proof obligations related to the control flows are proposed for verification. Traichaiyaporn et al. [11] combines Event-B with KAOS, which is a representative work of the goal oriented approaches of RE. They analyse the safety-related properties such as the consistency, relative completeness, domain evolution and deadlock-freeness to verify the correctness

of requirements. Butler et al. [13] combines UML models with Event-B. The correctness of requirements of a control system in UML-B is verified by checking the consistency through the refinement process and the equivalence between controller states and outputs. The ProR [15] approach enables the creation of a consistent system description from an initial set of requirements with tool support. It provides convenience of editing requirements and creating annotated links between requirements. Comparing with these approaches, our work takes account of the environment model in the first place according to the EBRE. The verification supported by Event-B guarantees that the system design never violate the restrictions from the environment in its further refinement process.

There are researches combining PF with Event-B. Traichaiyaporn et al. [16] defines the evolutionary framework based on PF to analyze the possibility of using Event-B to preserve the correctness of requirements in requirements evolution. However, the requirements are described in natural language, which makes a big gap from the formal model of Event-B. Besides, PF does not support to verify the restrictions from environment in comparision with EBRE. Gmehlich et al. [17] associate PF with Event-B through constructing a problem diagram capturing requirements before modeling in Event-B. The transformation from PF to Event-B is done manually, and the correspondence between PF and Event-B is not proved. They verify only the deadlock-freeness property, rather than the correctness of requirements. Comparing with them, our work defines the transformation from environment models to Event-B, and has the capability to verify the correctness of such requirements model.

For formal modeling and verification on composite structures, several works are devoted to event decomposition [18]. Butler et al. [19] propose event composition and decomposition theories for Event-B. Our approach utilizes the similar idea to deal with the composite structure in the scenario diagrams. In particular, the phenomena and variables are both shared in the scenario diagrams. We create events and variables to split them and correlate the values of decomposed variables with the original variable to verify the validity of refinement.

VI. Conclusion

In this paper, we proposed an approach to enhance the correctness of both requirements and design models through integrating EBRE and Event-B. We construct a well-defined framework to transform EBRE requirements models to Event-B models. We analyze the correspondence between the properties to be verified in EBRE and the POs generated from Event-B, which facilitates the usage of the formal verification technique in Event-B to verify the correctness of the requirements model. Besides, we develop the supporting plug-in tool for Rodin platform, which is beneficial to the automatic transformation from EBRE requirements model to Event-B model and verification for the correctness of requirements.

In the transformation rule proposed in this paper, the elements of EBRE and the elements of the Event-B are in one-to-one correspondence. Besides, when there are composition operations in the requirements model, the Event-B model will also have the corresponding transformation for the composition structure. The correctness of the transformation rules can be supported by (1) the correspondence between the informal semantics of EBRE and formal semantics of Event-B; (2) the consistency between the verification result from Event-B and analysis result from EBRE. We demonstrate the proposed approach on a simple running example of a light unit control software. The future work is to apply this approach on practical software development case to improve the quality of the software by integrate the requirements engineering and software design.

References

[1] A. Van Lamsweerde, "Goal-oriented requirements engineering: A guided tour," in *Proceedings fifth ieee international symposium on requirements engineering*. IEEE, 2001, pp. 249–262.

[2] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos, "Tropos: An agent-oriented software development methodology," *Autonomous Agents and Multi-Agent Systems*, vol. 8, no. 3, pp. 203–236, 2004.

[3] H. Mouratidis and P. Giorgini, "Secure tropos: a security-oriented extension of the tropos methodology," *International Journal of Software Engineering and Knowledge Engineering*, vol. 17, no. 02, pp. 285–309, 2007.

[4] C. Rolland, C. Souveyet, and C. B. Achour, "Guiding goal modeling using scenarios," *IEEE transactions on software engineering*, vol. 24, no. 12, pp. 1055–1071, 1998.

[5] M. Jackson, *Problem frames: analysing and structuring software development problems*. Addison-Wesley, 2001.

[6] X. Chen and Z. Jin, "Capturing requirements from expected interactions between software and its interactive environment: an ontology based approach," *International Journal of Software Engineering and Knowledge Engineering*, vol. 26, no. 01, pp. 15–39, 2016.

[7] Z. Jin, *Environment modeling-based requirements engineering for software intensive systems*. Morgan Kaufmann, 2018.

[8] Z. Jin, X. Chen, Z. Li, and Y. Yu, "Re4cps: requirements engineering for cyber-physical systems," in *2019 IEEE 27th International Requirements Engineering Conference (RE)*. IEEE, 2019, pp. 496–497.

[9] J. R. Abrial, *Modeling in Event-B: system and software engineering*. Cambridge University Press, 2010.

[10] N. Beauger, J. Bendisposto, M. Butler, A. Fürst, and A. Iliasov, "Deploy deliverable d23," 2010.

[11] K. Traichaiyaporn, "Modeling correct safety requirements using kaos and event-b," 2013.

[12] J. García-Duque, J. J. Pazos-Arias, M. López-Nores, Y. Blanco-Fernández, A. Fernández-Vilas, R. Díaz-Redondo, M. Ramos-Cabrer, and A. Gil-Solla, "Methodologies to evolve formal specifications through refinement and retrenchment in an analysis–revision cycle," *Requirements Engineering*, vol. 14, no. 3, pp. 129–153, 2009.

[13] C. Snook and M. Butler, "Uml-b: Formal modeling and design aided by uml," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 15, no. 1, pp. 92–122, 2006.

[14] "Requirements engineering for cyber-physical systems," [EB/OL], http://re4cps.org/home.

[15] M. Jastram, "The pror approach: Traceability of requirements and system descriptions," Ph.D. dissertation, 2012.

[16] K. Traichaiyaporn and T. Aoki, "Preserving correctness of requirements evolution through refinement in event-b," in *2013 20th Asia-Pacific Software Engineering Conference (APSEC)*, vol. 1, 2013, pp. 315–322.

[17] R. Gmehlich, K. Grau, S. Hallerstede, M. Leuschel, F. Lösch, and D. Plagge, "On fitting a formal method into practice," in *International Conference on Formal Engineering Methods*. Springer, 2011, pp. 195–210.

[18] T. S. Hoang, A. Iliasov, R. A. Silva, and W. Wei, "A survey on event-b decomposition," *Electronic Communications of the EASST*, vol. 46, 2011.

[19] R. Silva and M. Butler, "Shared event composition/decomposition in event-b," in *International Symposium on Formal Methods for Components and Objects*. Springer, 2010, pp. 122–141.