International Journal on Advanced Science Engineering Information Technology

Vol.12 (2022) No. 6
ISSN: 2088-5334

Keep All Objectives Satisfied (KAOS) to Event-B Models Transformation

Syahana Nur'Ain Saharudin [a,*], Mar Yah Said [a]

[a] *Department of Software Engineering and Information System, University Putra Malaysia (UPM), Serdang, 43400, Malaysia*
Corresponding author: [*] syahana.saharudin@gmail.com

***Abstract*— Requirements engineering is an important aspect of the software development methodology because it is the first phase in every software development. The usefulness of formal language in requirements is well-established to ensure consistency. However, the conversion from informal requirements to the formal specification phase is still challenging because it requires advanced skills and much practice. Due to this challenge, we improve the conversion and relationship of these two phases by capturing requirements using KAOS approach and writing the formal specification using Event-B language. KAOS approach allows modeling the requirements through goal hierarchies, whereas Event-B is a formal system-level modeling and analysis method. This work proposes model transformation rules from KAOS model to Event-B model, along with implementing the rules, and evaluates the proposed rules using Mine Pump Controller case study. We used a model-driven approach, specifically model-to-model transformation, to transform KAOS model to Event-B model. We modeled the case study into the KAOS model to obtain the source model for our model transformation and extend the existing KAOS meta-model by adding four new meta-classes to ensure the KAOS model can accommodate all Event-B components. Our proposed rules manage to generate an abstract Event-B model, and a set of proof obligations have been used to verify the correctness of the model. However, the designers must manually perform the transition between the generated outputs to the Event-B platform.**

***Keywords*— KAOS method; Event-B; goal-oriented requirements engineering; model-driven engineering; formal specification.**

Manuscript received 26 Jul. 2021; revised 24 Sep. 2021; accepted 30 Jun. 2022. Date of publication 31 Dec. 2022.

I. INTRODUCTION

Software development methodology consists of some phases, and each of these phases produces a number of artifacts that will be used in a later phase, such as requirements documents, architecture documents, test plans, and so forth. The first phase of every development process is the requirements phase, where the aim is to capture user's need, and later, its document will be used for generating other artifacts for later phases. Therefore, the requirements phase is a crucial step in every development [1], [2]. Requirements engineering focuses on eliciting, analyzing, specifying, and validating requirements related to a system that will be implemented [3]. If failures occur during requirements engineering phases, they can cause bad consequences. In order to overcome this problem, adopt a formal method during the specification and design phase. The specification is usually derived from requirements, and the derivation chain already exists partially [4], [5].

Using the formal method in establishing specification can prove the correctness of specification since formal methods contain proof theory that defines rules for deducing useful information. Formal methods have been used continuously in building complex system specifications [6]. The idea of the formal method is to refine an initial mathematical model until the final refinement model has sufficient information to be executed. Usually, we construct the initial model based on the requirements. However, there is a gap between textual requirements and initial formal specification [7]-[9]. Using initial formal specifications is not easy for the customers due to their lack of understanding of formal models. It is also difficult for the designers to link the formal models with the requirements. Because of this, the gap between requirements and formal specification phases grew larger, making the transition between them more difficult. Since formal methods are mainly used in complex systems, this problem can get worse because of many requirements. During this situation, the designers will try to perform pseudo-programming directly instead of building an abstract model [10].

Thus, we need to have an uninterrupted chain between requirements and specifications. The objective of this paper is to combine the requirements and specification phases by using KAOS [11], [12] and Event-B [13], [14] methods in order to explore the interconnection of the requirements model and formal method representation. KAOS is a goal-oriented approach in requirements engineering that allows the analyst to construct requirements models through goal hierarchies and derive requirements documents. It contains two refinement tactics, milestone-driven refinement, and decomposition-by-case refinement, and the goals can be refined using and/or refinements [15]. On the other hand, Event-B is a model-based formal method that provides formal semantics and availability of a toolset called Rodin [16], [17]. Similar to KAOS, it also contains a refinement feature and the process consist of evolving the abstract model into a concrete model [18].

Many works have been conducted to bridge the gap between requirements and specification phases using KAOS and Event-B. The approach proposed by Bicarregui et al. [19] used the KAOS goal model for analysis and elaboration of requirements, and their transformation focused on the leaf goals from the goal model. The leaf goal refers to a goal with no children or sub-goals. They introduced the notion of triggers in the event component.

Ponsard and Devroey [20] introduced a new approach by relying on KAOS agents and Event-B machines using UML-B and the decomposition technique in Event-B. Their approach involves creating the abstract machine and context to represent the entire KAOS object and agent models. They constructed several initial machines, where each initial machine represents one agent in the goal model.

The approach proposed by Matoussi *et al.* [10] concerned with the integration of Event-B model into KAOS goal model, where they expressed Event-B model using the concept of goal model. They proposed a model that could support the refinement mechanism of Event-B, but their model cannot support all Event-B components. In addition, this approach only focused on the event component of Event-B and did not consider other components of Event-B.

The latest approach by Fotso et *al.* [21] introduced the use of ontologies to enhance the SysML/KAOS model and more concerned with the translation of ontologies, or domain model, into B specification. They did not translate the model into Event-B specification. Their approach is different with ours because they emphasized the use of ontology in their transformation rules, which we did not use in our approach

For this project, we adopted an approach introduced by Matoussi *et al.* [10] and extend it to ensure all Event-B components are covered by proposing model transformation rules. In order to ensure the derivation from KAOS model to Event-B, we used a case study of Mine Pump Controller, which was taken from Ponsard and Devroey [20].

The contributions or specific objectives of our project are listed as follows:

- Model transformations rules for KAOS model to Event-B model are proposed.
- We implemented the model transformation rules using a model transformation language.
- We evaluated the correctness of the model transformation rules using a case study.

The content of this paper is organized as follows. Material and methods are described in Section II. We described the results of the transformation rules using the case study in Section III. In the last section, Section IV, is the conclusion for this project.

II. Material and Method

In this section, we present the rules that we used for the transformation of KAOS model to Event-B model. We also describe the technologies used to implement the transformation rules. In addition, we present the KAOS modelling for the case study that we used for evaluation purposes. Figure 1 shows the conceptual framework for our research.

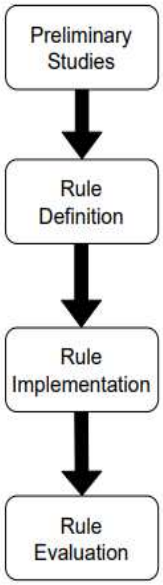


Fig. 1 Conceptual Framework of the Project

Our approach aims to bridge the gap between the requirements phase and formal specification by using KAOS and Event-B, where we want to derive the Event-B model from KAOS model. KAOS requirements model consists of four sub-models: goal, object, responsibility, and operating. Thus, for this approach, we used KAOS goal model to derive event component of Event-B while other components, such as context, variables and invariants, used KAOS object and responsibility model.

In the preliminary studies stage, we discover the recent methods or approaches we can use to define our model transformation rules. The work that has been chosen as the main reference for this project is Matoussi *et al.* [10], where they used KAOS goal model as the link to bridge the gap between requirements and formal specification. Thus, we used their concept, where each goal consists of target and current conditions. Matoussi *et al.* [10] used the current condition to represent guard component of Event-B event and the target condition to represent the action component of Event-B event.

The first contribution of this project is defining the transformation rules of KAOS model to Event-B model, which is the second phase in the conceptual framework. We used KAOS object model and responsibility model to enhance the approach proposed by Matoussi *et al.* [10] to

accommodate the Event-B components that they did not cater to. Thus, we made some adjustments on the existing meta-model by adding new meta-classes to assist us in deriving the Event-B context, and variable and invariant component of Event-B.

The second contribution is to implement the transformation rules using model transformation language, which is the third phase of our project methodology. Our project has chosen a model-driven engineering approach [22], [23]. We adopted model-to-model method to ensure the translation of KAOS model to Event-B model, and we used ATL language [24], [25] as the model transformation language.

Evaluating our transformation rules is our third contribution for this project. For this phase, we evaluated the rules using Mine Pump Controller case study [20]. First, we modelled the case study into KAOS model, and then we used the KAOS model to transform into Event-B model using our proposed transformation rules. We used the proof obligations of Event-B to verify the correctness of our output model. Event-B proof obligations are provided in Rodin toolset.

A. Rule Definition

The first objective of this project is to propose model transformation rules that can perform the translation of KAOS model to Event-B model. Thus, we took advantage of Eclipse's model management technologies, Eclipse Modelling Framework (EMF). We used model-to-model approach [26], and therefore, we used two meta-models, where one meta-model represents KAOS method and the other represents Event-B method.

For KAOS meta-model, we used the one introduced by Matoussi et al. [10] with new addition of meta-classes that are denoted by yellow in Figure 2. The four new meta-classes are needed to generate the Event-B context elements: sets, axioms and constants, and invariants and variable of Event-B machine. As for Event-B meta-model, we took the existing meta-model that had been provided by Rodin toolset.

As we can see, the meta-class *PackagedElement* composed of meta-class *Relationship*, meta-class *Goal* and meta-class *Object*. For meta-class *relationship*, it represents the goal refinement in the KAOS goal model and can be classified into three types: meta-classes *And, Or* and *Milestone*. As for meta-class *Goal*, it can be classified into two types: meta-class *AbstractGoal* and meta-class *ElementaryGoal*. Meta-class *AbstractGoal* concerns the most abstract goal in the goal model whereas meta-class ElementaryGoal concerns the sub-goals or children of the abstract goal.

The new meta-class *Object* represents the entity of the goal model that had been identified in the informal requirement. The bi-directional reference, such as central and sub, concerns with linking the object to its central object. If an object is a containment of the main object as the central object. Meta-class *Object* contains two new meta-classes. The first meta-class is meta-class *attribute*, which represents the properties of the object in the KAOS object model, whereas the new meta-class agent represents the agent responsible for the goal. Both meta-class attributes and meta-class agents consist of meta-class enumeration, representing the value in the object and responsibility models.

Based on the two meta-models, the mapping of the transformation model from source model, which is KAOS model, to target model, which is Event-B model, is shown in Table 1. As shown in the table, we can derive the variable and invariant components of Event-B machine using KAOS object model. The set, constant and axiom components, which are the components in Event-B context, are derived from the attributes in KAOS object model. We can also use KAOS agent model to derive Event-B context. The event component of Event-B can be obtained using the goals in KAOS goal model, and the variable component of Event-B machine can also be obtained from KAOS goal model.

TABLE I
MAPPING OF SOURCE MODEL TO TARGET MODEL

KAOS Meta-Model	Event-B Meta-Model
Object	Variable
	Invariant
Attribute	Set
	Constant
	Axiom
Agent	Set
	Constant
	Axiom
Goal	Event
	Variable

Table 2 describes the transformation rules we formulated for the translation of the two methods. We defined the rules based on the mapping of the two meta-models. Rule 1 with transforming meta-class *Object* into variable and invariant of Event-B machine. Rules 2 and 3 concern with the meta-class Attribute and Agent respectively are used to generate the Event-B context component. Rule 4 is used order to transforms the properties of meta-class *Goal* into event and variable of Event-B machine.

TABLE II
TRANSFORMATION RULES

Rule	Description
1(a)	Name of the object represents variable for Event-B machine
1(b)	Translating the object, its central object and its attributes into invariant of Event-B machine
2(a)	Name of the attribute represents the set of Event-B context
2(b)	The enumeration of attribute represents the constant of Event-B context
2(c)	Translating the name of attribute and its value into axiom of Event-B context
3(a)	Name of agent represents the set of Event-B context
3(b)	The enumeration of agent represents the constant of Event-B context
4(a)	Translating the name of the goal and taking the information obtained from Rule 4(b) and 4(c) in order to complete the event component in Event-B machine, and use the name of the parent goal to show the event refinement
4(b)	The target condition of goal represents the action of event
4(c)	The current condition of goal represents the guard of event
4(d)	The target condition of goal represents the variable of Event-B machine

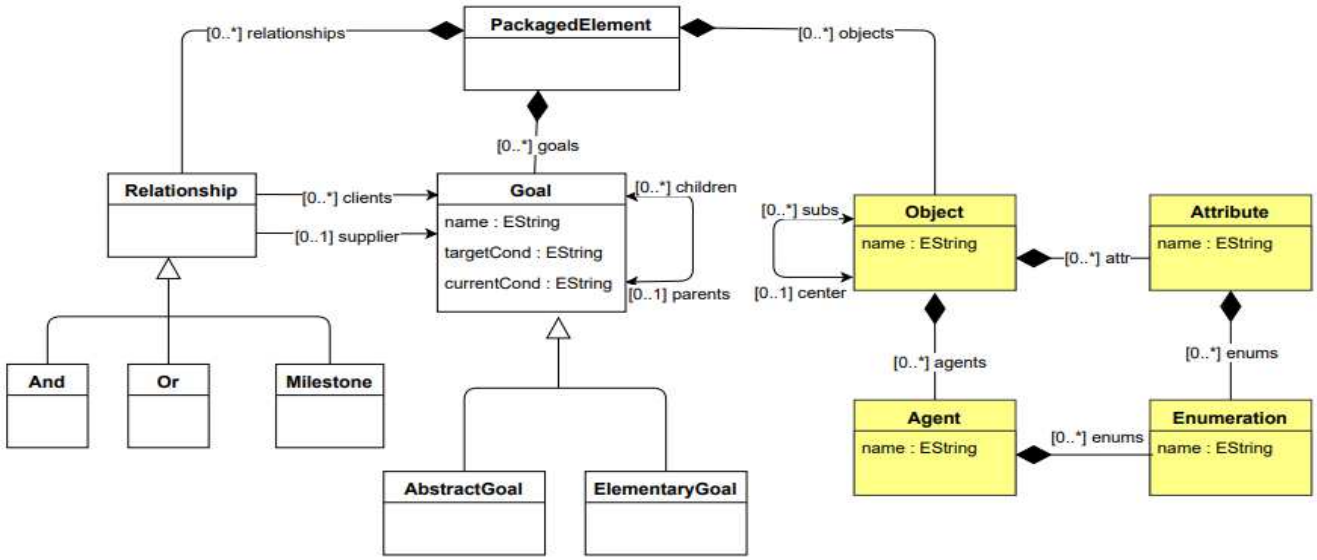


Fig. 2 Extension of KAOS Meta-Model

B. Rule Implementation

Model-driven engineering approach has been adopted in this project in order to transform one model to another model, where in this case we want to transform KAOS model to Event-B model. Our proposed rules were implemented using Eclipse platform based on model-to-model transformation. Atlas Transformation Language (ATL) has been chosen as the model transformation language since it has been widely used in model-driven engineering.

The basic concept of model transformation for our approach is shown in Figure 3. The transformation starts by taking a source model in the XML file and producing another XML model, the target model. The transformation is conducted by a transformation program, where, in this case, the rules that have been written in ATL language. The transformation program is also known as a model. Thus, the source, target, and transformation models must conform to their respective meta-models. Subsequently, these meta-models must conform to the meta-meta-model, which is the Meta-Object Facility (MOF).

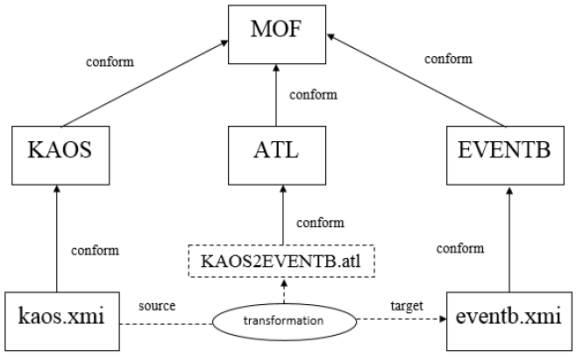


Fig. 3 Basic Concept of ATL Transformation

The transformation process occurred through declarative rules, which can call on auxiliary functions, also known as helpers in the syntax of ATL language. Based on the rules that have been explained in Table 2, we have derived four declarative rules. We used auxiliary functions to assist in extracting the information from the KAOS model, and we used declarative rules to transform the models.

C. KAOS Modelling

For evaluation purposes, we used a case study of Mine Pump Controller, which had been taken from Ponsard and Devroey [20]. It is a case study on the sump that is used to control the draining of water inside a mine. A sump is an area that collects the water entering the mine. The level of water is kept within bounds by a pump controller. Inside the mine are three sensors and two controllers that monitor the safety of the mine. The sensors are methane, high water, and low water, whereas the controllers are pump and alarm controllers. The requirements of the system are listed as follows:

- The pump should be switched on when water reaches high level to keep the mine dry and avoid flooding.
- The pump should be switched off when water reaches low level to avoid pump burning and damage.
- The pump should be switched off, and an alarm must be sounded when methane gas is detected inside the mine to avoid the explosion risk.

We construct the goal model for this case study using the goal-oriented requirements elaboration method, and Figure 4 illustrates the KAOS model for this case study. Based on the requirements listed above, we identify the preliminary goals, and from there, we elicit the requirements by asking WHY and HOW questions. Asking WHY questions assist in identifying high-level goals, and this process can be known as a bottom-up process. Meanwhile, asking HOW questions allow us to identify the sub-goals for the parent goal. Asking the two questions allow us to refine the goals until enough detail is acquired.

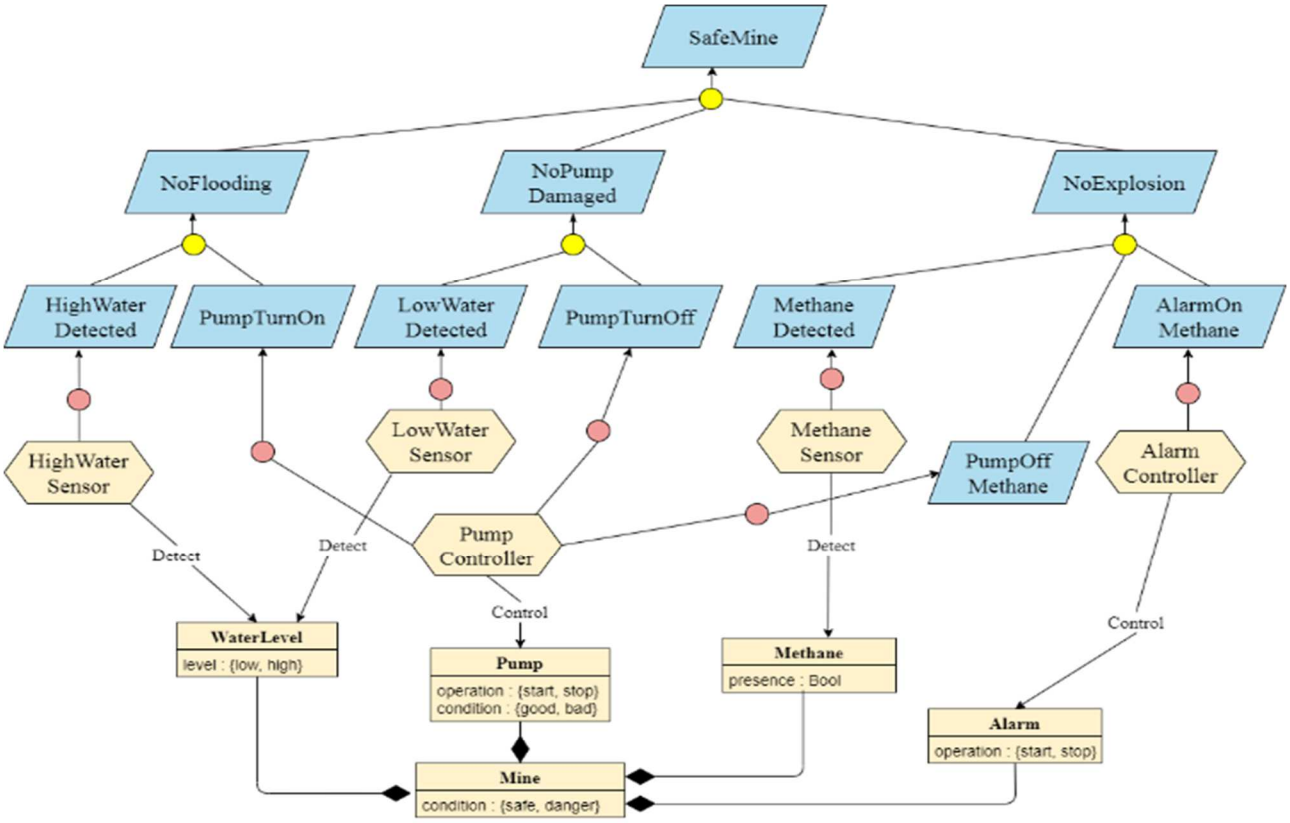


Fig. 4 KAOS Model of Mine Pump Controller Case Study

After the completion of refining the goals, we can assign an agent to the goals. In order to find the agents responsible for the goals, we asked WHO questions, where we want to identify the entity that plays the role of achieving the goal and is capable of monitoring and controlling the objects in the goal. We identified five agents interacting with the goals based on the problem statement mentioned in the first paragraph. The pump controller is responsible for controlling the pump by turning on/off according to water level. Alarm controller is responsible in controlling the alarm by switching on/off based on the presence of methane gas. Low water sensor is responsible in detecting the low water level inside the mine. High water sensor is responsible in detecting the high-water level inside the mine. Methane sensor is responsible in detecting the methane gas inside the mine. We also derived the objects and their attributes from the requirements to construct the object model. For this case study, we identified five objects or entities: mine, pump, alarm, methane, and water level. We can also assign agents to these identified objects.

III. Result and Discussion

This section describes the evaluation of our proposed translation rules using the Mine Pump Controller case study, along with the limitation of our approach.

A. Event-B Context Component

With the introduction of four new meta-classes, we can obtain Event-B context, which is missing from Matoussi *et al.* [10], our rules managed to generate the components for Event-B context. However, to ensure the transformation's correctness, the designer has to transfer the generated output into the Event-B platform manually. The Event-B context that our rules have transformed is shown in Figure 5.

Using Rule 2 and Rule 3, we obtain the sets, axioms and constants based on the meta-class *Attribute* and meta-class *Agent*. The designer manually adds the set MINE so that the variables can be declared as the subset of the set. As for other sets, we followed Rule 2(a), which took the name value in meta-class *attribute* and represented it as sets in Event-B context. Then, we followed Rule 2(b) to obtain the constants component. For example, in the object model, object Pump contains attribute operation, and the values for operation are start and stop. Using Rule 2(a) and Rule 2(b), the attribute operation becomes the set, and the value starts and stop become the constant in Event-B context. Subsequently, we used Rule 2(c), to obtain the axiom component, where it took the name and value of meta-class *attribute*. Using the same example, the axiom obtains from object Pump is *partition (OPERATION, {start}, {stop}).*

Rule 3 follows the same principle as Rule 2. The difference is Rule 3 takes the information from meta-class *agent*. Let us take the agent Pump Controller as an example. Following Rule 3(a), the name of the agent becomes the set component. However, for this case, we did not use its agent to represent the set component because all of the agents contain the same operation, which is on and off. Therefore, we took their enumeration to derive the constant component based on Rule 3(b).

Since Event-B does not allow duplication in terms of constant, we decided to combine the agents into one, leading to a new set *STATUS* that represents the operation of the

agents. For example, status refers to the state of the pump controller, whether it is turned on or off. Currently, there is no rule to assist us in deriving the set component regarding the agent that contains the same enumeration. Due to this, it is based on how the designer derives this component.

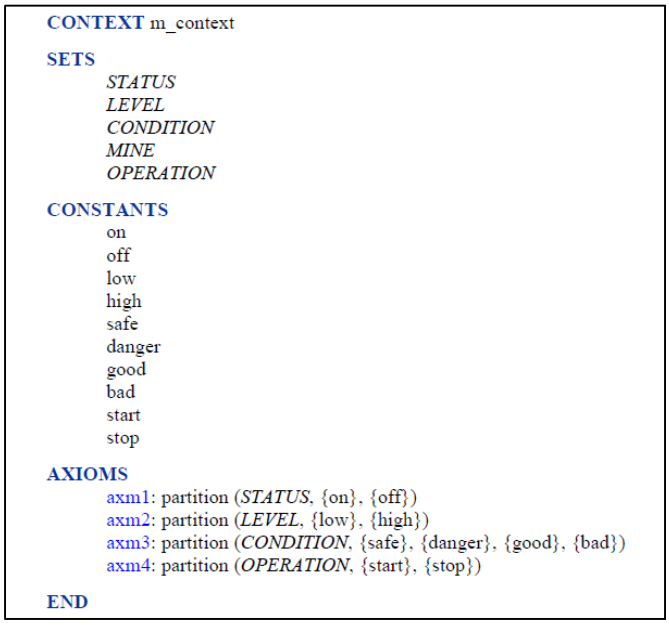


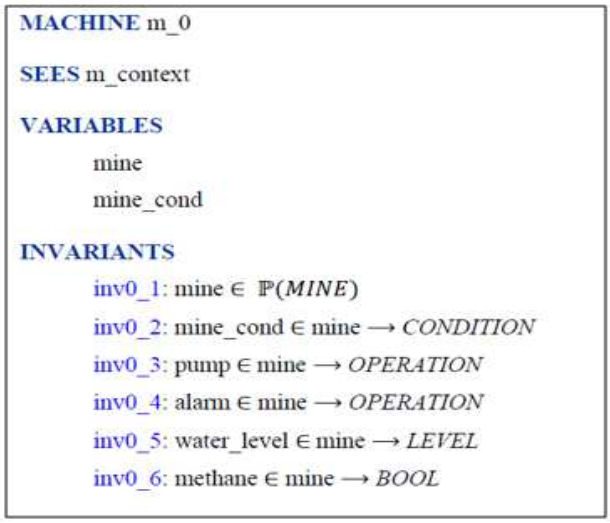
Fig. 5 Event-B Context

B. Event-B Abstract Machine

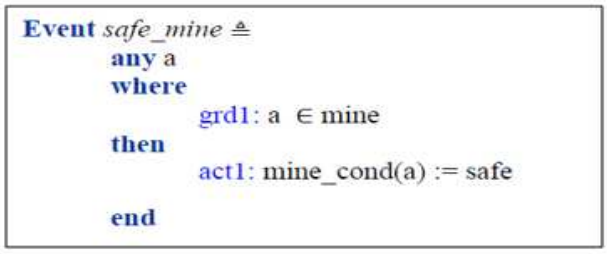
The addition of new meta-classes can also derive the variable and invariant component of Event-B machine as shown in Figure 6(a), following Rule 1 and Rule 4(d). Rule 1(a) is related to meta-class *Object*, where it took the object's name as the variable (*pump, alarm, methane, water level* and *mine*) while Rule 4(d) concerns with taking target condition of a goal as the variable (*mine_cond*). In order to obtain the invariant, we used Rule 1(b), where it used the name of the object, its central object and its attributes to represent the invariant.

Let us take object *WaterLevel* as an example. The central object for this object is object *Mine* and its attribute is *Level*. Therefore, we combine this three information and eventually create inv0_5, where it indicates that there is water level inside the mine and attribute Level indicates the type of water level (high, low). But it is different for the first invariant (inv0_1) because it is derived manually. For object mine, there is no central object, and this is because the object mine is the central object for other objects. Thus, we defined a user-defined type to give a type to mine, which is set MINE, and this process is done manually by the designer.

Rule 4 is used to transform the goal into event component in Event-B. Rule 4(a) used the goal's name to obtain the event's name. Let us take the goal *SafeMine* as an example. This goal is the abstract event in the abstract machine. Thus, the name of the abstract event is *SafeMine*, following Rule 4(a). The following rules in Rule 4, such as Rule 4(b) and Rule 4(c), are used to complete the event component. Rule 4(b) concerns taking the target condition as action of an event, whereas Rule 4(c) concerns taking the current condition as guard of an event. In order to obtain the guard and action for the event, the information is obtained through properties of the goal model. Table 3 shows the properties of goal *SafeMine* in the KAOS model.



(a)



(b)

Fig. 6 Event-B Abstract Machine: (a) Variable and Invariant Component (b) Abstract Event Component

TABLE III
PROPERTIES OF GOAL *SAFEMINE*

Properties	Value
Name	SafeMine
Parent	
Children	NoFlooding; NoPumpDamaged; NoExplosion
TargetCond	mine_cond = safe
CurrentCond	

Following Rule 4(b), the target condition (*mine_cond = safe*) represents the action of the event. As for current condition, there is no current condition for abstract goal because the abstract goal is the most abstract goal and general goal in the goal model. Therefore, the goal consists of target condition only. As for the parameter (*a*), we manually added the parameter in order to represent that the parameter is the type of central object. Figure 6(b) shows the abstract event *SafeMine* in Event-B specification

C. Event-B First Machine Refinement

Next is for the first machine refinement (*m_1*). Goal SafeMine consist of three sub-goals, which are goal NoFlooding, NoPumpDamaged and NoExplosion. Therefore, for the first machine refinement, it consists of three events. For example, the name of the sub-goal is NoFlooding. Therefore, NoFlooding is also the name of the event,

following Rule 4(a). This rule also completes the event component, where it calls the information obtained from Rule 4(b) and Rule 4(c) to get the guard and action of an event. Table 4 below shows the properties of goal model in our KAOS editor.

Rule 4(b) refers to the transformation of the target condition to action in an event. As shown in Table 4, the target condition for *NoFlooding* is the safe condition of flood (*flood_cond = safe*); thus, it represents the event's action. Rule 4(c) refers to transforming the current condition to guard of an event. For example, the current condition for NoFlooding is the operation of the pump, where the pump has to start operating (pump = start) to ensure no flooding occurs in the mine. As for event refinement, Rule 4(a) is used to obtain the parent goal. The rule takes the name of the parent goal, for example the parent goal for *NoFlooding* is *SafeMine*, and later show that the event *NoFlooding* is refining the abstract event *SafeMine*. The first guard (*grd1*) and first action (*act1)* of this event is kept from the abstract event because this event refines and extends the abstract event. Figure 7 shows the event in Event-B specification for the sub-goal *NoFlooding*.

TABLE IV
PROPERTIES OF GOAL *NOFLOODING*

Properties	Value
Name	NoFlooding
Parent	SafeMine
Children	HighWaterDetected; PumpTurnOn
TargetCond	flood_cond = safe
CurrentCond	pump = start

Event *no_flooding* ≙

refines *safe_mine*
any *a*
where

grd1: a ∈ mine
grd2: pump(a) = start

then

act1: mine_cond(a) := safe
act2: flood_cond(a) := safe

Fig. 7 Excerpt of Event in First Machine Refinement

D. Event-B Second Machine Refinement

For the second machine refinement (*m_2*), the derivation of event also used Rule 4. For example, the sub-goals or children for goal *NoFlooding* are goal *HighWaterDetected* and *PumpTurnOn*. Let us take goal *PumpTurnOn* as an example. Following Rule 4(a), the goal's name becomes the event's name. Rule 4(a) is also used for event refinement, where it takes the name of the parent goal to show the refinement. The parent goal for this goal is goal *NoFlooding*. Thus, the rule will show that event *PumpTurnOn* is refining the event *NoFlooding*. The target condition for this goal is to turn on the pump controller (*pump_ctrl = on*) and the current conditions for the goal are high water level (*water_level = high*) and the initial state of pump controller (*pump_ctrl = off*). Using Rule 4(b), the target condition becomes the event's action, and using Rule 4(c), the current conditions become the guard of event. Similar to event *NoFlooding*, the first guard (*grd1*), second guard (*grd2*), first action (*act1*) and second action *(act2*) are kept from the first refinement event. Table 5 shows the properties for goal *PumpTurnOn* whereas Figure 9 shows the event component in second machine refinement.

TABLE V
PROPERTIES OF GOAL *PUMPTURNON*

Properties	Value
Name	PumpTurnOn
Parent	NoFlooding
Children	
TargetCond	pump_ctrl = on
CurrentCond	water_level = high; pump_ctrl = off; methane = false

Event *pump_turn_on* ≙

refines *no_flooding*
any *a*
where

grd1: a ∈ mine
grd2: pump(a) = start
grd3: pump_ctrl(a) = off
grd4: water_level(a) = high
grd5: methane(a) = FALSE

then

act1: mine_cond(a) := safe
act2: flood_cond(a) := safe
act3: pump_ctrl(a) := on

end

Fig. 8 Excerpt of Event in Second Machine Refinement

E. Event-B Proof Obligations

After the generation of Event-B components using KAOS model, we verified the correctness of the Event-B model. The model is proved using a set of proof obligations (POs) that had been generated by Rodin platform. Proof obligations are used to ensure the consistency of a certain property in formal specification [26]. If we can prove the proof obligations, then that property in the specification is consistent. The main properties for checking the correctness of Event-B model are refinement between models, well-definedness of expressions and invariant preservation [27].

The statistics of proof obligations for our Event-B model is described in Table 6. The total refers to the total number of POs that had been performed on the model. Auto PO refers to the number of proof obligations that are automatically discharged by the provers whereas manual PO refers to the number of PO that are manually discharged. Two proof obligations are done manually, and these POs are done by re-running the proof obligation using external prover.

TABLE VI
STATISTICS OF PROOF OBLIGATIONS

Machine	Auto PO	Manual PO	Total PO
m_0	1	1	2
m_1	12	1	13
m_2	26	0	26

F. Limitations

Even though our rules managed to transform KAOS model to Event-B model, there are still some limitations because the implementation of our rules can only generate the XML file for Event-B model and therefore need the help of the designer to manually convert the XML file into Event-B specification in order to verify the correctness of the model. We did not manage to develop a tool that could support our proposed rules due to time constraints. In addition, our approach focused only on functional requirements, unlike Matoussi *et al.* [10], whose approach covered both functional and non-functional requirements.

Currently, the generated output from the model transformation rules can only give the designers the general layout for constructing the system's Event-B specification. The output can assist the designer in classifying the information from the goal model and object model into Event-B components. However, the designer might need to double-check the translation and do some editing on the translation done by the rules to accommodate the Event-B specification and ensure the model's correctness.

In addition, our rules did not cater to machine refinement. Our rules only consist of two machines: abstract machine and first machine refinement. Therefore, for the subsequent machine refinement, the designer must perform manually per the goal model. Our proposed rules are still in an early stage of development and require some improvements, especially on ensuring the machine refinement and integration of ATL codes with the Rodin platform to make the model transformation automatic.

IV. Conclusion

Our approach focused on the transition from the requirements phase to the specification phase, where the main objective is to bridge the gap between the two phases, and we used a model-driven method to achieve the objective. The literature review that we performed to facilitate the transition gave some insight into the limitation of the existing approaches. The translation concept proposed by Matoussi *et al.* [10] was adopted in our approach, where we mapped the *Achieve* goal of KAOS model with event of Event-B machine. Then, we used the current and target condition in the goal to represent the guard and action in Event-B event component. We made some adjustments to the existing meta-model by adding four new meta-classes so that we can accommodate other components of Event-B, such as Event-B context. We used model-to-model transformation to perform the translation of two models using model transformation technologies, such as EMF and ATL. We chose Mine Pump Controller case study to model its requirements into KAOS model and evaluate the correctness of our proposed model transformation rules.

Currently, our approach is semi-automated because we need the help of Event-B designer to transform the generated output into Event-B specification manually. This process can be simple for small systems, but it can be tiresome when it involves large numbers of requirements. Thus, it is beneficial to have an automated tool for transforming the KAOS model to Event-B.

Since our rules did not cater to machine refinement, the ATL rules only defined the two machines using an entry point rule, which did not use any information from the source model. There is no specific rule for machine refinement. In order to cater to the limitation of machine refinement, we can use meta-class relationships using the goal's relationship. Not only that, by using this meta-class, we can make our event refinement more systematic.

The evaluation of our proposed rules is currently done using one case study. We might use other case studies to ensure the correctness of our transformation model. Also, the modeling tool for KAOS, such as Objectiver [28], is not an open-source tool. It is hard to obtain the XML file for our source model. Thus, we made our KAOS editor using EMF to gain the input file. Due to time constraints, we could not construct an open source KAOS editor; therefore, we can consider this as our future work. An open-source KAOS editor can ensure the possibility of making an automatic version of the model transformation.

Acknowledgment

We thank the Faculty of Computer Science and Information Technology (FCSIT) of the University Putra Malaysia (UPM) for financial assistance.

References

[1] J. Horkoff, F. B. Aydemir, E. Cardoso, T. Li, A. Mate, E. Paja, M. Salnitri, L. Piras, J. Mylopoulous and P. Giorgini, "Goal-oriented requirements engineering: An extended systematic mapping study," *Req. Eng.,* vol. 4, no. 2, pp. 133-160. June. 2019, doi: 10.1007/s00766-017-0280-z.

[2] R. Kasauli, E. Knauss, J. Horkoff, G. Liebel and F. G. de Oliveira, "Requirements engineering challenges and practices in large-scale agile system development," *J. Sys. Soft.,* vol. 172. Elsevier Ltd, p. 110851, Feb. 2021, doi: 10.1016/j.jss.2020.110851.

[3] M. Tukur, S. Umar and J. Hassine, "Requirement engineering challenges: A systematic mapping study on the academic and the industrial perspective," *Arabian J. Sci. Eng.,* vol. 46, no. 4, pp. 3723-3748, Apr. 2021, doi: 10.1007/s13369-020-05159.

[4] H. Kaiya, N. Yoshioka, H. Washizaki, T. Okubo, A. Hazeyama, S. Ogata and T. Tanaka, "Eliciting requirements for improving users' behavior using transparency," in *Comm. Comp. Inf. Sci.,* 2018*,* vol. 809, pp. 41-56, doi: 10.1007/978-981-10-7796-8_4.

[5] M. M. Awan, F. Azam, M. W. Anwar and Y. Rasheed, "Formal requirements specification: Z notation meta model facilitating model to model transformation," in *ICSIE 2020 – Proc. 2020 9th Int. Conf. Soft. Info. Eng.,* Nov. 2020, pp. 61-66, doi: 10.1145/3436829.3436845.

[6] M. Gleirscher, S. Foster and J. Woodcock, "New opportunities for integrated formal methods," *ACM Computing Surveys,* vol. 52, no. 6, pp. 1-36, Oct. 2019, doi: 10.1145/3357231.

[7] M. Ozkaya, "Do the informal and formal software modelling notations satisfy practitioners for software architecture modelling?" *Info. Soft. Tech.,* vol. 95, pp. 15-33, March. 2018, doi: 10.1016/j.infsof.2017.10.008.

[8] E. Alkhammash, "Formal modelling of OWL ontologies-based requirements for the development of safe and secure smart city systems," *Soft Computing,* vol. 24, no. 15, pp. 11095-11108, Feb. 2020, doi: 10.1007/s00500-020-04688-z.

[9] E. Osama, M. Abdelsalam and A. Khedr, "The effect of requirements quality and requirements volatility on the success of information systems projects," *Int. J. Adv. Comp. Sci. App.,* vol. 11, no. 9, Oct. 2020, doi: 10.14569/IJACSA.2020.0110950.

[10] A. Matoussi, F. Gervais and R. Laleau, "A goal-based approach to guide the design of an abstract Event-B specification," in *16th IEEE Int. Conf. Eng. Complex Comp. Sys.,* 2011, pp. 139-148, doi: 10.1109/ICECCS.2011.21.

[11] E. Souza and A. Moreira, "Deriving services from KAOS models," In *Proc. ACM Symposium on Applied Computing,* 2018, pp. 1308-1315, doi: 10.1145/3167132.3167273.

[12] A. Cailliau and A. van Lamsweerde, "Runtime monitoring and resolution of probabilistic obstacles to system goals," *ACM Transactions on Autonomous and Adaptive Systems,* vol. 14, no. 1, pp. 1-40, Sept. 2019, doi: 10.1145/3337800.

[13] J. R. Abrial, *Modelling in Event-B – System and software engineering,* 1st ed. Cambridge: Cambridge University Press, 2010, pp. 1-586, doi: 10.1017/CBO9781139195881.

[14] C. Zhu, M. Butler, and C. Cirstea, "Trace semantics and refinement patterns for real-time properties Event-B models," *Science of Computer Programming,* vol. 197. Elsevier Ltd, p. 102513, Oct. 2020, doi: 10.1016/j.scico.2020.102513.

[15] N. Ulfat-Bunyadi, N. G. Mohammadi, R. Wirtz and M. Heisel, "Systematic refinement of softgoals using a combination of KAOS goal models and problem diagrams," in *Comm. Comp. Inf. Sci.,* 2019*,* vol. 1077, pp. 150-172, doi: 10.1007/978-3-030-29157-0_7.

[16] J. R. Abrial, M. Butler, S. Hallerstede, T. S. Hoang, F. Mehta and L. Voisin, "Rodin: An open toolset for modelling and reasoning in Event-B," *Int. J. Soft. Tools for Tech. Transfer,* vol. 12, no. 6, pp. 447-466, Apr. 2010, doi: 10.1007/s10009-010-0145-y.

[17] P. Andre, C. Attiogbe and A. Lanoix, "A tool-assisted method for the systematic construction of critical embedded systems using Event-B," *Comp. Sci. Info. Sys.,* vol. 17, no. 1, pp. 315-338, Jan. 2020, doi: 10.2298/CSIS190501042A.

[18] A. Mammar, M. Frappier, S. J. T. Fotso, and R. Laleau, "A formal refinement-based analysis of the hybrid ERTMS/ETCS level 3 standard," *Int. J. Soft. Tools for Tech. Transfer,* vol. 22, no. 3, pp. 333-347, June 2020, doi: 10.1007/s10009-019-00543-1.

[19] J. Bicarregui, A. Arenas, B. Aziz, P. Massonet and C. Ponsard, "Towards modelling obligations in Event-B," in *Lecture Notes in Computer Science,* vol. 5238, 2008, pp. 181-194, doi: 10.1007/978-3-540-87603-8_15.

[20] C. Ponsard and X. Devroey, "Generating high-level Event-B system models from KAOS requirements models," in *XXIXeme Congres INFORSID 2011 – 29th Conf. INFORSID*, 2011, pp. 317-332.

[21] S. J. T. Fotso, M. Frappier, R. Laleau and A. Mammar, "Modelling the hybrid ERTMS/ETCS level standard using a formal requirement engineering approach," *Int. J. Soft. Tools for Tech. Transfer,* vol. 22, no. 3, pp. 349-363, June 2020, doi: 10.1007/s10009-019-00542-2.

[22] D. Akdur, V. Garousi, and O. Demirors, "A survey on modelling and model-driven engineering practices in the embedded software industry," *J. Sys. Arch.,* vol. 91, pp. 62-82, Oct. 2018, doi: 10.1016/j.sysarc.2018.09.007.

[23] A. Bucchiarone, J. Cabot, R. F. Paige and A. Pierantonio, "Grand challenges in model-driven engineering: An analysis of the state of the research," *Soft. and Sys. Modelling,* vol. 19, no. 1, pp. 5-13, Jan. 2020, doi: 10.1007/s10270-019-00773-6.

[24] E. Richa, E. Borde and L. Pautet, "Translation of ATL to AGT and application to code generator for Simulink," *Soft. and Sys. Modelling,* vol. 18, no. 1, pp. 321-344, Feb 2019, doi: 10.1007/s10270-017-0607-8.

[25] S. Gotz and M. Tichy, "Investigating the origins of complexity and expressiveness in ATL transformations," *J. Obj. Tech.,* vol. 19, no. 2, pp. 1-21, July 2020, doi: 10.5381/jot.2020.19.2.a12.

[26] A. P. F. Magalhaes, A. M. S. Andrade and R. S. P. Maciel, "Model driven transformation development (MDTD): An approach for developing model to model transformation," *Info. Soft. Tech.,* vol. 114, pp. 55-76, Oct. 2019, doi: 10.1016/j.infsof.2019.06.004.

[27] A. S. A. Hadad, C. Ma and A. A. O. Ahmed, "Formal verification of AADL models by Event-B," *IEEE Access,* vol. 8, pp. 72814-72834, Apr. 2020, doi: 10.1109/ACCESS.2020.2987972.

[28] K. Morris, C. Snook, T. S. Hoang, G. Hulette, R. Armstrong and M. Butler, "Formal verification of run-to-completion style statecharts using event-b," in *Comm. Comp. Info. Sci.,* 2020, vol. 1269, pp. 311-325, doi: 10.1007/978-3-030-59155-7_24.

[29] *Objectiver* (2007). [Online]. Available: http://www.objectiver.com/