A Domain Experts Centric Approach to Formal Requirements Modeling and V&V of Embedded Control Software

Weikai Miao *, Qianqian Yan, Yihao Huang, Jincao Feng, Hanyue Zheng
Shanghai Key Lab for Trustworthy Computing
East China Normal University
Shanghai, China
wkmiao@sei.ecnu.edu.cn *

***Abstract*—Formal method is a promising solution for precise software requirements modeling and V&V (Validation and Verification). However, domain experts are suffering from using complex mathematics formal notations to precisely describe their domain specific software requirements. Meanwhile, the lack of systematic engineering methodologies that can effectively encompass precise requirements modeling and rigorous requirements V&V makes the application of formal methods in industry still a big challenge. To tackle this challenge, in this paper, we present a domain experts centric approach to the formal requirements modeling and V&V in the domain of embedded control software. The major advancements of the approach are: 1) a domain-specific and systematic engineering approach to the formal requirements specification construction and 2) scenario-based requirements validation and verification requirements technique. Specifically, the approach offers a domain-specific template for formal specification construction through a three-step specification evolution process. For formal requirements V&V, diagrams are derived from formal specification and domain experts' concerned scenarios can be checked based on the diagrams. These modeling and V&V technologies are coherently incorporated in the approach and fully automated by a supporting tool. We have applied the approach real software projects of our industrial partners. The experimental results show that it significantly facilitates the formal modeling and V&V in industry.**

Keywords—formal methods, requirements specification, requirements V&V

I. INTRODUCTION

For safety-critical control systems, such as the railway transportation system, aerospace control system and nuclear plant control system, whether their embedded control software can perform the expected functions correctly affects the human's safety. Therefore, how to ensure the correctness of embedded control software attracts the research efforts from both the academic and the industry [1]–[5].

It is well recognized that a formal requirements specification can effectively contribute to the quality of the software system [1, 2, 6]. Some industrial standards such as the DO-333 for aircraft software also strongly recommended the application of formal methods [5]. By building a formal specification, expected functions are precisely defined. Therefore, rigorous V&V (i.e., **validation** and **verification**) can be performed [1, 7]. Software can then be developed based on a firm foundation.

For industrial practitioners, however, it is still a great challenge to apply formal requirements modeling and V&V in real industrial software projects [6, 7]. One major problem is the gap between domain experts and requirements engineers. Specifically, the problem is twofold:

1) lack of way to build formal requirements specification

Traditional way of requirements specification construction is using natural language to describe system functions by the software engineers based on the communications with the domain experts. However, software engineers may not fully understand particular domain knowledge. On the other hand, domain experts are almost impossible to understand and use formal notations to describe software requirements. Because most domain experts do not have solid background of discrete mathematics and learning complicated mathematics notations and proof skills are time-consuming and difficult. Therefore, domain knowledge may not be sufficiently and precisely described in the requirements specification.

2) lack of V&V from the domain experts' perspective

Requirements V&V are usually carried out by software engineers, dealing with specific functions and variables. From domain experts' view, however, they do not understand tedious variables or functions. By contrast, they are more interested in whether the expected scenarios are correctly defined by the requirements specification. That is, the V&V should be performed from domain experts' perspective rather than only from software engineers' view. Traditional formal verification focuses on formal proof for detecting logical errors, e.g., inconsistency, but it is difficult for domain experts. Meanwhile, such formal verification is usually performed for system design but falls short in requirements analysis.

Both domain experts and requirements engineers are demanding appropriate engineering approaches that can encompass formal specification construction from scratch and effective requirements V&V. Such approaches should offer systematic way for specification evolution from initial informal and insufficient requirements to the final formal specification. Meanwhile, such approaches need also provide effective techniques for checking whether the formal requirements specification correctly describe the domain experts' expectations.

To this end, we propose an evolutionary formal requirements modeling approach for the requirements specification construction and V&V of embedded control software. Specifically, we first summarized common features of embedded control software and designed a group of requirements specification template. For a systematic requirements modeling, requirements are first written in the informal specification. When the informal specification is confirmed, it is then transformed into a semi-formal specification in which data structures are precisely defined while the pre- and post-conditions of each function can still be

* Corresponding author.

2640-0715/19/$31.00 ©2019 IEEE
DOI 10.1109/APSEC48747.2019.00012

informal. Such semi-formal specification can easily be used for requirements reformulation. Finally, the specification evolves to the formal specification in which all the data structures and pre- and post-conditions of functions are completely defined. As for the V&V, various diagrams are generated from the formal specification. Based on the intuitive diagrams, domain experts can check whether the specification conforms to their expected scenarios through static review and dynamic analysis. This approach and our supporting tool have been applied by our industrial collaborators such as the CASCO Co., Ltd. (the largest railway signal producer of China) and the ACAE Co., Ltd. (the main aircraft engine producer of China). The application results show that the approach significantly improve their requirements modeling and analysis.

The remainder of this paper is organized as follows. In Section 2, we give the overview of the related work. Section 3 gives the framework of the approach and the background of a running example. Section 4 and Section 5 describe the evolutionary specification construction and requirements V&V, respectively. Section 6 reports our case studies and experimental results. Section 7 summarizes the paper and point our future research topics.

II. RELATED WORK

Formal methods have been developed over decades. For example, the Z [8] and the Event-B [2] are traditional ones. Domain specific language (DSL) based requirements modeling approaches are also proposed [9, 10]. However, traditional formal methods and DSL based approaches fall short in offering a systematic engineering process of specification construction for large-scale industrial projects. Industrial practitioners are struggling with detailed maths notations rather than focusing on the domain problems. StateChart is widely used in industrial software modeling and analysis [11], and Matlab/Simulink provides Stateflow [12]. Based on the StateChart and Stateflow, various traditional analysis and formal V&V activities can be performed. The BIP model is also used for component-based software modeling and validation [13, 14]. But these notations and approaches are more suitable for system design. The SOFL is a formal engineering method, which offers an evolutionary engineering process of formal specification construction and corresponding V&V technologies [15]–[17]. We also share the same idea of the evolutionary specification construction. Moreover, our approach is more domain specific and includes a domain expert's centric V&V technique. In work [18], the authors present an approach for requirements validation by using scenario questions. In [15], an inspection method based on condition data flow diagrams is proposed. These are static V&V methods based on review techniques. In the work [19], the authors present a group of analysis techniques for architectural models described in EAST-ADL, including the simulation of EAST-ADL model in Simulink. The BIP method also use model checking for system validation [13, 14]. The authors of [20] present an algorithm for the statistical model checking of Markov chains with respect to unbounded temporal properties. But model checking is slightly difficult for domain experts. Compared with these approaches, our requirements V&V focuses on the domain experts' view and facilitating both static and dynamic requirements analysis. The major advancements of our approach are domain-specific and domain experts centric.

III. FRAMEWORK OF THE APPROACH

In this section, we first give an overview of the approach and then describe a running example.

A. The Approach

The approach consists of two major techniques: the evolutionary requirements modeling and the V&V, which is described in Figure 1.

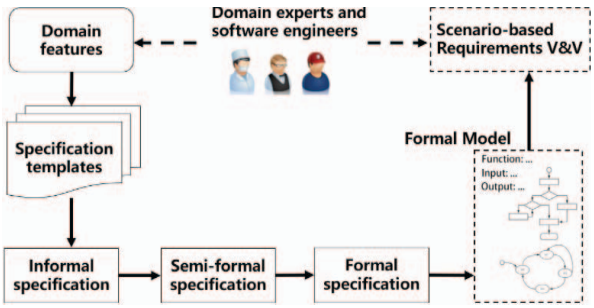

Fig. 1. Framework of the Approach

Requirements modeling should take domain knowledge into consideration. In this approach, we first extract some common and typical features of the embedded control software through extensive collaboration with the domain experts. Such features, for example, periodic signal and mode transitions, are summarized and organized into structured templates (e.g., templates for aircraft control software and railway signal control software). In the template, we define some sections for writing corresponding requirements..

Based on the template, requirements are then documented using informal language guided by the pre-defined keywords and sections. In general, for the domain of embedded control software, functions and data dictionary may necessarily be described. Such informal specification can be easily understood by both software engineers and domain experts. When the informal specification is confirmed, then it is transformed into the semi-formal specification. In the semi-formal specification, all the data structures are precisely define. That is, all the input and output variables involved in the functions are precisely defined. However, the pre- and post-condition that specifies the relations between the input and output variables are still informal. In this specification, system architectures can be easily reformulated and such specification can still be useful for communication between domain experts and software engineers. When the semi-formal specification is determined, then it is finally transformed into the formal specification in which all the variables and functions are formally defined.

Traditional formal V&V are difficult for domain experts to be involved. In our approach, formal specification is transformed into a requirements model for machine processing. Diagrams such as state transition diagrams, mode relation diagrams and variable dependency diagrams are automatically generated from the model for review and formal analysis. Such V&V focuses on checking whether the domain experts' expected scenarios of the system is correctly and sufficiently described by the formal specification. That is, this V&V is performed from the domain experts' perspective and allow the experts to easily be involved in the requirements analysis.

B. A Running Example

We use a "Motion Control" subsystem of an ATP (Automatic Tran Protection) software system (provided by the CASCO Co., Ltd.) as the running example. ATP is a control software for railway trains. Since an ATP is a large-scale system, in this paper, we will use the Motion Control subsystem to explain how the approach works in practice. The Motion Control subsystem is in charge of monitoring and controlling the motions of a train by monitoring the speed and various sensor signals and giving control signals. For the domain experts, their task is to precisely define the functions and check whether their expected scenarios of the system are correctly and sufficiently described by the formal specification.

IV. EVOLUTIONARY SPECIFICATION CONSTRUCTION

A. Specification Templates Design

Although each software system has its unique features, there are still certain common and typical features shared by all the systems in the same domain. These features are actually the essential elements for requirements modeling. Requirements templates should be designed to support the representation of these domain features. Therefore, domain experts can easily participate in the specification construction since such templates reflect their domain knowledge and can guide them to describe system functions.

We focus on the domain of embedded control software in our research. According to our long-term researches and collaborations with industrial partners, some important features of embedded control software are summarized below.

- periodic control: system functions are executed based on certain periodic signal. For these periodic-driven systems, periodic signals need to be explicitly defined.
- global variables: control software usually includes various global variables
- hierarchical structures: organizations of functions needs to be clearly specified to represent system architectures

These features actually are the fundamental domain knowledge in the domain of embedded control software. Suppose we design an appropriate mechanism for describing such features, then domain experts can easily be involve in the requirements modeling. Our way is to design a set of requirements specification templates for guiding the specification construction. Figure 2 shows some examples of specification templates.

Template 1 is the specification template for mode transition style systems such as the engine control software systems of aircraft. Temple 2 is the specification template for structured functions systems such as software of railway

Template 1

Mode: Em_Stop
{
Pre-condition: ES=URG
Procedure:
Period: 5(ms)
{**Call** （Function X）}
Transitions:
Target: Ground
Guard: Np<NT
Func: X
Input: SensorFlt
Output: ValidSpeed
Task: if(SensorFlt==…)
…
}

Template 2

Functions: Motion
Input: x
Output: y
Pre-condition: x>0
Post-condition: y=x+1

Data Dic
int x
int y
int z

Invariants:
z>x+y

Fig. 2. Examples of the Specification Template

signal control. All the domain features are organized as sections and keywords in the templates. For instance, in template 2, system requirements can be described as functions, data dictionaryand invariants. Each function has its input and output variables that should be specified in the data dictionary. The pre- and post-condition specifies the relations between the input and the output variables. Guiding by the template, domain experts can follow their traditional way of system requirements elicitation without dealing with complicated maths details.

B. Specification Construction

Based on the template, specification construction is an evolutionary process including the following three steps: informal, semi-formal and formal specification constructions.

1) Informal Specification Construction: Requirements are rough and insufficient at the initial stage of specification construction. In this stage, frequent communications between domain experts and software engineers are very important. To this end, it is more appropriate to write specification using informal languages rather than any formal notation. In our approach, after selecting a proper template, both the domain experts and the software engineers start to write down requirements in the specification using certain natural language. Figure 3 shows a part of the informal specification of the running example.

Function
SWRQ-0200
ATP should manage the train movement using a motion estimation model.
The maximum and minimum motion shall be estimated based on different states.
The allowed states are **COASTING**, **SKIDDING**, **SIDING** and **BRAKING**.
The state transfers from **COASTING** to **BRAKING** when…
Input: Odometer Information…
Output: Motion Monitor Ended Signal…
Data Dic
…
Invariants

Fig. 3. Examples of the Informal Specification

The requirement numbered SWRQ-0200 is a motion control function, which can monitor and control the motion states of a train. The input of the function includes the data read from the odometer signals. After the computation based on a motion control model, the function gives the corresponding state transition signals.

2) Semi-formal Specification Construction: When the informal specification is finished, it is then transformed into a semiformal specification. The major task of building a semi-formal specification is twofold: 1) formally defining all the data structures involved in the specification and 2) reformulating requirements into independent functions.

By formally defining the data structures and clarifying the requirements into functions, the specification is a structured document with precise interface definitions. Note that the preand post-conditions of each function can still be informal. Such transformation from informal specification to semiformal specification is mainly achieved by manual work based on the understanding of domain knowledge and requirements, since clarifying the informal requirements is actually an intellectual work that depends on the understanding of the intended functionalities of the target software system. Figure 4 is the semi-formal specification of the previous informal one.

Function 0206
Input: MoOvEsState: StateEnum, WhMaSp: SpeedUnit
Output: StSlSp: SpeedUnit
If system is initialized or state is COASTING or BRAKING, then the sliding speed should be zero.
If system starts sliding, the sliding speed should equal to the wheel speed.
Otherwise, it should be the value of the previous period.

Function 0210
Input: StBrMoMa: MoUnit, WhMaMo: MoUnit
Output: MaMoDuBrOrSl: SpeedUnit
If system braking motion value is great then 0, maximum speed of motion during braking should be calculated according to Formula Motion.

Fig. 4. Examples of the Semi-formal Specification

The requirement numbered SWRQ-0200 is decomposed into several functions. The input and output variables of each function is precisely defined while the pre- and post-conditions are still in informal. For instance, variable MoOvEsState is an enumeration type variable and the type of variable WhMaSp is SpeedUnit that has been defined in the data dictionary. The semi-formal specification is a structure requirements specification that keeps the balance between precision and readability. It is suitable for quick review and simple validation. When the experts and the software engineers confirm the specification, it is then processed into a formal specification.

3) Formal Specification Construction: The major task of building a formal specification is to completely transform all the components in the semi-formal specification into formal definitions. We have defined a light-weight formal notation for the domain of embedded control software. Due to the space limitation, we do not give the detailed syntax and semantics of the notation in this paper. The notation can be referred to [10]. In general, the formal notation is similar to the style of Python language, which can be easily mastered by both domain experts and software engineers of our industrial partners. Figure 5 is the example of the formal specification.

Function 0206
Input: MoOvEsState: StateEnum, WhMaSp: SpeedUnit
Output: StSlSp: SpeedUnit
if (Initialization
*or (*MoOvEsState*(k) == COASTING)*
*or (*MoOvEsState*(k) == BRAKING))*
StSlSp= *0*
else:
*if (*MoOvEsState*(k-1) == COASTING)* StSlSp = WhMaSp *(k-1)*
else: StSlSp = StSlSp*(k-1)*
End;

Function 0210
Input: StBrMoMa: MoUnit, WhMaMo: MoUnit
Output: MaMoDuBrOrSl: SpeedUnit
*if (*StBrMoMa*(k-1) > 0)*
MaMoDuBrOrSl*(k)*
= min*((*MaMoDuBrOrSl*(k-1)* + WhMaMo*(k)), 0)*
else:
*if (*StBrMoMa*(k-1) < 0)*
MaMoDuBraOrSl*(k)*= max*((*MaMoDuBrOrSl*(k-1)* + WhMaMo*(k)), 0)*
else:
MaMoDuBraOrSl = *0*
End;

Fig. 5. Examples of the Formal Specification

In the formal specification, the pre- and post-conditions are formally specified to precisely describe the relations between the input and the output variables of each function. For instance, the pre- and post-conditions are formally defined as if-else conditions. To support the representation of periodic control signal, we introduce a signal variable k. For instance, in the specification MaMoDuBrOrSl(k) denotes the value of variable MaMoDuBrOrSl in the current time period while MaMoDuBrOrSl(k-1) denotes the value in the previous period. For simplicity, k can be omitted if the user try to represent the variable value in the current period.

By following the three-step modeling process, requirements are gradually formalized in the specification. Since the requirements modeling process is carried out by human, errors are inevitably in the specification. To check whether the specification correctly represent the expected functions and scenarios, rigorous validation and verification should be performed.

V. REQUIREMENTS VALIDATION AND VERIFICATION

Domain experts focus on whether their concerned scenarios are correctly and sufficient described by the formal specification. That is, the V&V should be domain experts centric rather than a “black-box” to the domain experts. Meanwhile, the V&V should be automated for the sake of efficiency. To help the domain experts participate in the requirements V&V, one important point is to give an intuitive representation of the specification. To this end, we propose a domain experts centric requirements V&V technique. Formal specification is automatically transformed into a formal model. Diagrams are then generated from the model for guiding the rigorous review and scenario-based V&V.

A. Model Generation

To transform the formal specification into an executable model for diagrams generation and dynamic verification, a compiler is needed. The compiler contains two parts: a parser and a model generator. The parser takes the formal

specification as input and then convert it into an AST (Abstract Syntax Tree). The AST is then given to the model generator. The model generator can traverse the AST and transform it into the formal model.

Our supporting tool uses the ANTLR (ANother Tool for Language Recognition) [21] as a component for implementing the parser. From a given grammar, ANTLR generates a parser that can build and analyzes AST. After getting the syntax of our formal language, ANTLR returns a lexer and a parser for processing the formal specification.

The model generator is constructed to transform the AST into the formal model for V&V. The model is equivalent to the formal specification but it is used for machine reading. To obtain information from the given AST, the model generator uses depth-first search to traverse each child node of the tree.

We take the function 0210 in the formal specification as an example. Fig. 6 lists its pre- and post-condition.

```
if (StBrMoMa(k-1) > 0):
    MaMoDuBrOrSl(k) = min((MaMoDuBrOrSl(k-1) + WhMaMo(k)), 0)
else:
    if (StBrMoMa(k-1) < 0):
        MaMoDuBrOrSl(k) = max((MaMoDuBrOrSl(k-1) + WhMaMo(k)), 0)
    else:
        MaMoDuBrOrSl(k) = 0
```

Fig. 6. Segment of Formal Specification.

The function mainly describes the assignment of variable MaMoDuBrOrSl. The function is converted into an AST and then transformed as the formal model shown in Fig. 7. The model just serves as a machine readable model that is equivalent to the formal specification. By analyzing the model, we then can generate various diagrams for requirements V&V. Algorithm 1 in Fig. 8 describes how a state transition diagram is automatically derived from the requirements model. The algorithm takes the model as a CFG (Control Flow Graph) as its input and transform the control flows into a state transition diagram. Similarly, other diagrams can also be derived by following the same principle, i.e., extracting the CFG structures.

B. Scenario-based Requirements V&V

In our approach, we generate various diagrams from the requirements model. The diagrams includes the mode transition diagram, state transition diagrams, variable dependency diagrams and module diagrams. Based on the diagrams, we can perform both static and dynamic V&V. Due to the space limitation, in this paper, we focus on the most commonly used state transition diagram and its corresponding V&V, since embedded control software systems are mostly state-transition systems. Other diagram based V&V techniques can be referred to our previous work [10]. For a concise illustration of the V&V, we use the ”Motion Control” subsystem of our running example. Part of its formal specification can be referred to the description of Function 0206 in Fig.5.

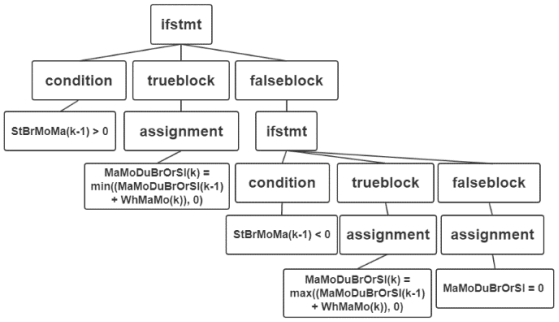

Fig. 7. Segment of Formal Specification in Model.

Algorithm 1 StateTransition(StateTran)

```
1:  Input : CFG,StateVar,ConditionSet(ConSet),Source(S)
2:  Output : StateTransRel
3:  if CFG.Type == ChoiceStmt then
4:    for all Expression(Expr) in CFG.Condition do
5:      if Expr is AssignExpr && Expr.LeftVar == StateVar
        then
6:        Source = Expr.RightVar
7:      else
8:        add Expr to TempCondition
9:      end if
10:   end for
11:   TrueCondition = ConSet union TempCondition
12:   StateTran(CFG.TrueBlock,StateVar,TrueCondition,S)
13:   FalseCondition = ConSet union !TempCondition,
14:   StateTran(CFG.FalseBlock,StateVar,FalseCondition,S)
15: else if CFG.Type == CompoundStmt then
16:   for cfg from CFG.begin to CFG.end do
17:     StateTran(cfg,StateVar,ConSet,S)
18:   end for
19: else if CFG.Type == SimpleStmt then
20:   if CFG is AssignExpr && CFG.LeftVar == StateVar
      then
21:     add (S,ConSet,CFG.RightVar) to StateTransRel
22:   end if
23: end if
24: return StateTransRel
```

Fig. 8. Algorithm of the StateTransition(StateTran).

1) Static Requirements V&V: Static V&V is mainly performed by reviewing the diagrams. By reviewing the intuitive diagrams, domain experts can quickly find some incorrect descriptions of the requirements. Figure 9 is the derived state transition diagram of the formal specification of the ”Motion Control” subsystem of our running example.

The state transition diagram visualizes the functions of the Motion Control subsystem. It has four states and eight transition conditions. For example, the train can transit from state “COASTING” to state “BRAKING”. Based on the domain knowledge, experts can find potential errors through inspecting the diagram. For instance, after carefully inspecting the state transition diagram, the experts may find that system should be able to directly transit from state “SKIDDING” to state “BRAKING” in case of emergency according to their domain knowledge. Therefore, the missing transition of the requirements specification is detected.

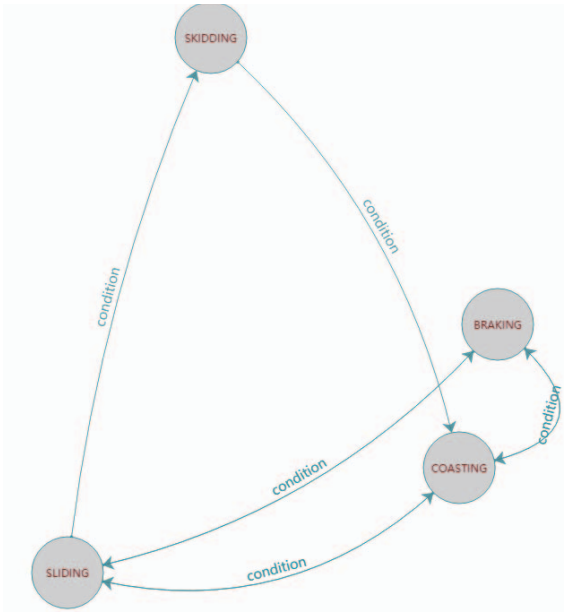

Fig. 9. State Transition Diagram of "Motion Control".

Such static review can really help validate the requirements . The reason is that requirements validation, to a large extent, depends on the experts' experience and domain knowledge. On the other hand, such review only checks the requirements statically. Potential errors cannot sufficiently be detected. To this end, we adopt the scenario-based dynamic V&V.

2) Dynamic Requirements V&V: Scenario is a domain expert view of the system. Our goal of the dynamic V&V is to check whether the requirements specification correctly and sufficiently defines the concerned scenarios. In particular, since embedded control software is typically time period driven, some behaviors can only be observed via "running" the model. From the domain experts' view of requirements V&V, there are some common and typical scenarios are required to be verified. For a state or a mode transition based control software, two scenarios need to be checked.

1) **unambiguous transition:** for each state or system mode, only one transition is valid at a specific time period

2) **state reachability:** each state can be reached from a given other state

The above two scenarios are very common in embedded control software. The first scenario requires that for any state that has various transition condition to others, only one condition is activated at any time period. Otherwise, the system transition behavior will be non-deterministic. The second one ensures that each state is accessible from a given state.

We first consider the first scenario. To ensure the unambiguous transition of each state, the transition conditions to other states should be mutual exclusive. The basic idea of this V&V consists of three steps.

1) collecting the transition conditions

2) generating pairs of transition conditions

3) using the Z3 solver [22] to check whether each pair of transition conditions can be satisfied.

TABLE I. TRANSITION CONDITIONS OF STATE SLIDING

Condition
C1: ((State(k - 1) == SLIDING) && ((OdSt(k) is INVALID) || (MaMoOdSiCh(k) == True)))
C2: ((State(k - 1) == SLIDING) && (OdSt(k) is INITIALIZED) && (not MaMoOdSiCh(k)) && (TiInSl(k - 1) ≤ ATSlTi) && (0 < (abs(StSlSp(k - 1)) + TiInSl(k - 1) * ATSdStAc)) && ((abs(StSlSp(k - 1)) + TiInSl(k - 1) * ATSdStAc) < abs(WhMaSp(k))) && (SlEn(k) == True))
C3: (State(k - 1) is SLIDING && OdSt(k) is INITIALIZED && not MaMoOdSiCh(k) && (TiInSl(k - 1) > ATSlTi || (StSlSp(k - 1) + TiInSl(k - 1) * ATSdStAc) ≤ 0 || (((StSlSp(k - 1) + TiInSl(k - 1) * ATSdStAc) ≥ WhMaSp(k)) && SlEx(k))))

Obviously, for a state which has m transition conditions to other states, it has $\frac{m(m-1)}{2}$ pairs of conditions to be checked. The checking of conditions pairs is actually a constraint solving problem. If there exists a solution that satisfies a condition pair, we say the two conditions are not exclusive. That is, the state transitions are incorrect. We use the Z3 Solver for constraints solving. If the Z3 Solver returns Status. UNSATISFIABLE for the input pair, then the two conditions which construct the pair are mutually exclusive. Otherwise, the conditions are wrong, which means that the state transitions of the concerned state are incorrect.

We use our running example for the illustration. Tab. I lists the transition conditions from state SLIDING to other states. We use C1, C2, C3 to represent these conditions. "C1 && C2", "C1 && C3" and "C2 && C3" are collected and put into the Z3 Solver. For these three input pairs, Z3 Solver returns "Status.UNSATISFIABLE". So it is clear that the transition conditions of state SLIDING are mutually exclusive.

For checking the second scenario, we need to "execute" the model and then observe whether the state transitions conform to the expectation. For a certain state T, the basic idea is to check its reachability is the following steps.

1) selecting a given state S;

2) setting the time periods to be checked, e.g., 500 time periods;

3) domain experts give the input values of the variables involved in the transition conditions to be checked;

4) if the requirements model can reach state **T** before the end of last time period, then we say state **T** is reachable from state S.

Note that the most important point is the generation of input values for each transition condition to be checked. This generation depends on the domain experts because only the experts know which values are useful and necessary for the requirements V&V. And this is why we named our approach

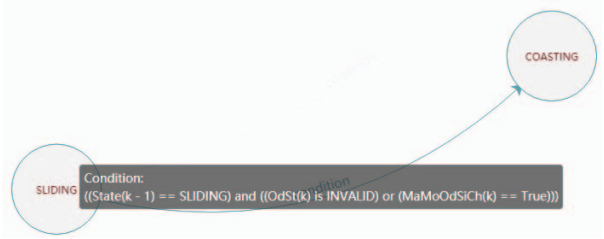

Fig. 10. Transition Condition from state SLIDING to state COASTING.

as “domain experts centric”.

We use the same example to explain how to validate and verify the second scenario. A part of the state transition diagram of our running example is shown in Fig. 9. Moving mouse pointer to the ”condition” label on any of the directed edges, specific transition condition will exist.

“((State(k - 1) == SLIDING) and ((OdSt(k) is INVALID) or (MaMoOdSiCh(k) == True)))” is the transition condition from state SLIDING to state COASTING. ”
In Fig. 8, we set the state ”SLIDING” as the starting state and state “BRAKING” as target state to verify the reachability of state “BRAKING”. Then we input the values of variables of 50 cycles. Based on the domain knowledge of train control, the domain experts give the following input values of the variables, which is shown in Table 2.

The specific means of each variable and its value in this table is quite difficult for software engineers to understand. However, the domain experts are familiar with these contents.

TABLE II. VALUES OF INPUT VARIABLES FOR DYNAMIC V&V

Variable Name	Value	Variable Name	Value
MaTrMo(k-1)	3	MiTrMo	0.1
StBrMoMa(k-1)	7	StBrMoMi	0.3
WhMaSp(k-1)	314	WhMiMo	0.5
TiInSl(k-1)	3	WhMaMo	100
OdSt	INVALID	MaMoDuBrOrSl	0
MaMoOdSiCh	True	ATOdMiDiAfSeCh	0.7

Here we just focus on the results. At the end of the 9th time periods, system transits from current state “SLIDING” to “COASTING” based on the inputs. However, after the 15th time periods, the transition condition from “SLIDING” to “COASTING” is evaluated to be false. Therefore, the state “BRAKING” is unreachable from state “SLIDING” under this instance of scenario.

We may find that there exists other two paths SLIDING, SKIDDING, COASTING, BRAKING and SLIDING, BRAKING that can reach state BRAKING from state SLIDING. However, according to the given input, transition conditions of these two paths are false. That is, under the specific instance of scenario given by Table 2, the target state cannot be reached. In this case, the domain experts determines that the requirements specification cannot conform to the expected scenarios. In our approach, the construction of scenarios depends on the domain expert’s understanding of some industrial standards and experience, which involves a lot of domain knowledge. However, to fully ensure the completeness is a difficult problem, which needs more researches in the future.

VI. CASE STUDIES AND EXPERIMENTAL RESULTS

To evaluate the feasibility of the approach and the usability of our supporting tool, we have adopted the approach and our tool in several industrial partners’ projects in recent years. These enterprises includes the CASCO (the largest railway signal producer of China) and the AECC Commercial Aircraft Engine Co., Ltd (the largest aircraft engine producer of China).

We applied our requirements modeling and V&V in two ATP projects of the CASCO. For simplicity, we call these two projects as P1 and P2. A group of requirements engineers joined this case study, 3 engineers with the background of train control as domain experts and 1 software engineer. Two PhD students also joined the group as software researchers. The formal specifications of P1 and P2 consisted of 450 and 445 functions, respectively. For P1 and P2, we generated 5 and 8 state transition diagrams for V&V, respectively. As a result, 3 and 1 ambiguous errors were detected from P1 and P2, respectively. And from P2, we detected 2 unreachable transitions. These errors were not found through traditional specification V&V. Apart from the state transition diagrams, we also generated 300 variable dependency diagrams for each project for rigorous review.

The approach also improves the efficiency of requirements modeling and V&V. Compared with traditional formal specification construction and V&V by manual work, time costs have been reduced from 3 months to less than 2 months. Such improvement may differ because of various factors, e.g., the user’s experience or project contents, nevertheless, our approach facilitates an effective way of formal specification construction in which more potential errors can be discovered.

The approach is also applied in an aircraft engine control software. The specification of the engine control software is composed of 10 system mode (i.e., can be regarded as system states) and 152 functions. Through the collaboration of 1 domain experts and 2 software engineers, it took 2.5 months for the group to build a formal requirements specification. After the requirements V&V, 3 errors were detected from the mode transition diagrams.

In summary, the experimental results of our case studies have convinced us that the approach can really help the domain experts in the field of embedded control software to improve the requirements modeling and analysis. It makes it efficient for practitioners to construct formal requirements specifications instead of using informal specification for requirements analysis. Based on the precise formal specification, more errors are automatically detected. From the perspective of time cost, building a formal specification may takes time. As the domain expert’s feedback, one important advantage of this approach is the ”consideration of domain-specific knowledge and features”. Compared with traditional formal methods, this approach makes it possible to involve domain experts in the entire requirements modeling and V&V process. Another important issue is the ”engineering process” that incorporates the evolutionary requirements modeling and the scenario-based V&V in a coherent manner. The approach tells the practitioner ”what to do” and ”how to do” in building the formal specification rather than only giving complicated notations and formal proof rules.

In our approach, we have spent some time in defining the specification templates since we need to collaborate with domain experts and understand domain features. On average, it needs 2-3 months to design an appropriate template. Currently, this is the necessary cost for applying formal methods in industrial for requirements modeling and analysis. We also hope that we may find more efficient way, such as natural language processing techniques, to improve the template processing.

VII. CONCLUSION

In this paper, we present a domain experts centric formal requirements modeling approach to embedded control software. The requirements specification construction is guided by a template that is developed based on the domain knowledge and features. Requirements specification is gradually constructed through building the informal, semi-formal and formal specification. The formal specification is then transformed into a formal model for generating various diagrams. Based on the diagrams, rigorous V&V is performed from the domain experts' perspective. Experts can check whether the concerned scenarios are correctly and sufficiently defined by the specification. The approach is applied in real aircraft engine control software and automatic train protection (train control) projects of our industrial partners, the AECC Commercial Aircraft Engine Co., LTD and the CASCO Co., Ltd. The experimental results demonstrate the feasibility and advantages of our approach.

The approach is a part of our long-term research plan. We are considering more intelligent technologies to deal with the problem of formal modeling, for example, using the natural language processing for automated specification derivation from user requirements. To achieve more efficient V&V is also one of our research topics in the future.

ACKNOWLEDGMENT

This work is supported by the NSFCs of China (No. 61872144 and No. 61872146).

.

REFERENCES

[1] A. Hall, "Realising the benefits of formal methods." J. UCS, vol. 13, no. 5, pp. 669–678, 2007.

[2] J. R. Abrial, Modeling in Event-B - System and Software Engineering. Cambridge University Press, 2013.

[3] M. Bozzano, A. Cimatti, A. F. Pires, D. Jones, G. Kimberly, T. Petri, R. Robinson, and S. Tonetta, "Formal design and safety analysis of air6110 wheel brake system," in International Conference on Computer Aided Verification. Springer, 2015, pp. 518–535.

[4] M. Bozzano, A. Cimatti, J.-P. Katoen, P. Katsaros, K. Mokos, V. Y. Nguyen, T. Noll, B. Postma, and M. Roveri, "Spacecraft early design validation using formal methods," Reliability engineering & system safety, vol. 132, pp. 20–35, 2014.

[5] S. RTCA, "Do-333 formal methods supplement to do-178c and do-278a," Tech. Rep., 2011.

[6] L. E. G. Martins and T. Gorschek, "Requirements engineering for safetycritical systems: An interview study with industry practitioners," IEEE Transactions on Software Engineering, 2018.

[7] S. Liu, Formal Engineering for Industrial Software Development: Using the SOFL Method. Springer Science & Business Media, 2013.

[8] J. Jacky, The way of Z: practical programming with formal methods. Cambridge University Press, 1997.

[9] Z. Wang, G. Pu, J. Li, Y. Chen, Y. Zhao, M. Chen, B. Gu, M. Yang, and J. He, "A novel requirement analysis approach for periodic control systems," Frontiers of Computer Science, vol. 7, no. 2, pp. 214–235, 2013.

[10] W. Miao, G. Pu, Y. Yao, T. Su, D. Bao, Y. Liu, S. Chen, and K. Xiong "Automated requirements validation for atp software via specification review and testing," in International Conference on Formal Engineering Methods. Springer, 2016, pp. 26–40.

[11] D. Harel, "Statecharts: A visual formalism for complex systems," Science of computer programming, vol. 8, no. 3, pp. 231–274, 1987.

[12] (2019) The mathworks: Stateflow and stateflow coder, users guide. Available:www.mathworks.com/help/releases/R13sp2/pdfdoc/stateflow/sfug.pdf

[13] E. Stachtiari, A. Mavridou, P. Katsaros, S. Bliudze, and J. Sifakis, "Earlyvalidation of system requirements and design through correctness-byconstruction," Journal of Systems and Software, vol. 145, pp. 52–78, 2018.

[14] A. Mavridou, E. Baranov, S. Bliudze, and J. Sifakis, "Configuration logics: Modeling architecture styles," Journal of Logical and Algebraic Methods in Programming, vol. 86, no. 1, pp. 2–29, 2017.

[15] M. Li and S. Liu, "Integrating animation-based inspection into formal design specification construction for reliable software systems," IEEE transactions on reliability, vol. 65, no. 1, pp. 88–106, 2015.

[16] S. Liu, A. J. Offutt, C. Ho-Stuart, Y. Sun, and M. Ohba, "Sofl: A formal engineering methodology for industrial applications," IEEE Transactions on Software Engineering, vol. 24, no. 1, pp. 24–45, 1998.

[17] W. Miao and S. Liu, "A formal engineering framework for service-based software modeling," IEEE Transactions on Services Computing, vol. 6, no. 4, pp. 536–550, 2012.

[18] D. Aceituna, H. Do, and S.-W. Lee, "Sqˆ(2) e: An approach to requirements validation with scenario question," in 2010 Asia Pacific Software Engineering Conference. IEEE, 2010, pp. 33–42.

[19] R. Marinescu, H. Kaijser, M. Mikucionis, C. Seceleanu, H. L ˇ onn, and ¨A. David, "Analyzing industrial architectural models by simulation and model-checking," in International Workshop on Formal Techniques for Safety-Critical Systems. Springer, 2014, pp. 189–205.

[20] P. Daca, T. A. Henzinger, J. Kˇret´ınsky, and T. Petrov, "Faster statistical ` model checking for unbounded temporal properties," ACM Transactions on Computational Logic (TOCL), vol. 18, no. 2, p. 12, 2017.

[21] T. Parr. (2014) Antlr. [Online]. Available: https://www.antlr.org/

[22] (2019) Z3 solver. [Online]. Available: https://rise4fun.com/z3/tutorial