Hierarchical Model Exploration for Exposing Off-Nominal Behaviors

Kaushik Madala
Computer Science and Engineering
University of North Texas
kaushik.madala@my.unt.edu

Hyunsook Do
Computer Science and Engineering
University of North Texas
hyunsook.do@unt.edu

Daniel Aceituna
DISTek Integration, Inc.
Daniel.Aceituna@distek.com

***Abstract*—Finding and addressing off-nominal behaviors in the later phases of the software development life cycle is costly and time consuming. In our previous work, we proposed the causal component model (CCM), a semi-automated approach that can expose off-nominal behaviors in requirements. The empirical study with CCM showed promising results, but CCM has limitations with scalability. To reduce the scalability issues, in particular because of hierarchical relationships, we propose an enhanced causal component model that uses hierarchical model exploration. The results of our case study indicate that the proposed approach reduces the state/rule explosion problem and detects off-nominal behaviors.**

I. Introduction

Embedded systems provide enormous benefits[1]. However, most embedded systems are susceptible to off-nominal behaviors (ONBs) [2], [3] that can lead to system failure. ONBs are unexpected or unintended behaviors [2], [3]. ONBs can be caused by various reasons such as a user performing an operation or triggering an event without following the correct procedure, two components of system being in contradictory states of operation, or environmental factors affecting the system's operations. For example, in a microwave oven, an unexpected behavior would be the oven cooking while the door is open.

To date, various approaches for detecting and correcting ONBs have been proposed, but many of these approaches require product implementation [4], [5], [6] and only a few of them are at requirements level [2], [7]. However, catching and correcting defects in the later phase of software development is usually more time consuming and costly than catching them in an early phase (e.g., requirements engineering phase) [8], [9].

To address this lack, our previous work proposed the causal component model (CCM) [7], which generates a model from the rules created from natural language requirements and performs a model analysis against undesired behaviors. While empirical studies with the CCM approach showed promising results, the approach has limitations with scalability. CCM suffers from state explosion and rule explosion problems, and cannot handle sub-states without performing state flattening [10]. State flattening removes hierarchical relationships and results in atomic states, which can affect dependencies in a hierarchy and lead to a large number of state transitions in CCM. To handle these limitations, we improved the CCM approach and implemented the enhanced causal component model (ECCM) by considering hierarchical model exploration to reduce the state and rule explosion problem [11]. The main intention of proposed approach is to reduce scalability issues because of sub-states. The main idea of our approach is that, as the component state transitions among super-states do not require the details of sub-states, when they are analyzed separately, the number of system-states and system transition rules generated are small, and thus it simplifies the model analysis. To evaluate our proposed approach, we performed a case study using the pacemaker requirements document [12]. Our results indicate that the ECCM can reduce the state and rule explosion problem and detect ONBs.

The rest of paper is organized as follows. Section II provides background information on the CCM, and Section III presents the proposed approach. Sections IV and V present a case study and discuss the results of the study. Sections VI and VII present related work, and conclusions respectively.

II. Background

The causal component model (CCM) [7] is generated from requirements to expose ONBs. It models the system behavior using three model elements: component, state, and transition condition. A component is an entity that is a part of the system's composition that can change states or trigger a state change in other components. In the case of embedded systems, components can be both hardware and associated software functional units. For example, in a robot system, components might include the motor, gyroscope and ultrasonic sensor. Each component has its own set of possible states, such as a motor in a robot having off and on states. These states are component states. Components' states can be grouped together to form a system state. For example, in a robot system, while motor in its 'on' state is referred as component state, the state of robot when motor is 'on', gyroscope is 'on' and ultrasonic sensor is 'on' represents the system state of robot . Transition condition is the event or condition that trigger for a state transition. These transition conditions can be environmental (i.e, a person or environmental factor triggers events), or system related (i.e, component states trigger state transitions). These model elements are identified in natural language requirements and transition rules are written. The transition rules are in the form

of mapping function *Transition Condition: Component(current state) → Component(next state)*.

The component states are combined into system states, converting component state transition rules to system state transition rules. The ONBs that could be caused by non-recoverable and undesired states are found by checking the model against undesired rules. Once the ONBs are determined, the stakeholders can address the problems. The process is repeated until the outcome satisfies the stakeholders. More detailed discussion of CCM can be found in [7].

III. The Proposed Approach: ECCM

To address the limitations of the CCM, we propose the enhanced causal component model (ECCM). Figure 1 illustrates an overview of the proposed approach. The ovals represent the processes and the rectangles represent the software artifacts (inputs and outputs of the processes). The numbers just outside the ovals indicate step numbers. Each step is detailed as follows: *1: Convert natural language requirements to ECCM specifications:* The first step is to convert natural language requirements into ECCM specifications. To convert the requirements to the input specification to generate ECCM, we identify model elements: components, components' states, and transition conditions. The explanation on these model elements can be found in Section II. The input specification of ECCM is component state transition rules. A transition rule of ECCM is in the form of *Transition Condition: Component(current state) → Component(next state)*. After identifying the model elements, the transition rules are created. Because ECCM considers sub-states differently than CCM approach (where states are flattened), we represent sub-state as parent state followed by dot ('.') followed by sub-state. For example, a component motor has 'off' and 'on' states and 'on' state has 'accelerate' and 'decelerate' states. 'On' state of motor is represented as Motor(on), where as 'accelerate' sub-state of 'on' state is represented as Motor(on.accelerate). If there are multiple levels of sub-states, then the representation will be the outer most state followed by dot ('.') followed by its sub-state followed by dot followed by its sub-state and the process is repeated until the inner-most sub-state which needs to be represented is found.

2.) Partition rules into rule sets: The rules generated may result in too many system states, and consequently the number of system transition rules created may also be increased. In order to reduce state explosion and rule explosion, we partition the rules into rule sets based on hierarchy levels of sub-states and components (see step 2 of Figure 1).

The advantages of partitioning are that we analyze only a part of system at a time and if the users are not domain experts or non-technical experts and need to understand the level of operation at higher or abstract level, then they need to go through only one rule set (first rule set) which represents the abstract representation of system. In addition, the generation on rule sets based on hierarchy eases the analysis process of technical experts. They can consider only part of system which has issues and analyze it easily. Because our approach uses graph-based technique for analyzing off-nominal behaviors (ONBs), the dependencies between the components are not lost.

Algorithm 1: Generation of Rule Sets

input : A set of components in system $C = \{C_1, C_2, C_3, ..., C_n\}$
A set of states at level 0 for each component
maxlevel, maximum level of sub-states
Sets of sub-states for each component at level 1, 2, 3, ..., maxlevel
Rules R = $\{R_1, R_2, R_3, R_4, ..., R_w\}$
output : Rulesets

```
1  rulesetnum = 0;
2  for currentlevel ← 0 to maxlevel do
3      if currentlevel == 0 then
4          foreach Rule R_i in R do
5              if states in R_i ∈ any of component-states at level 0 then
6                  Rulesets[rulesetnum] ← Rulesets[rulesetnum] ∪ R_i;
7          rulesetnum ← rulesetnum+1;
8      else
9          foreach Component C_i in C do
10             if Component C_i has sub-states at level = currentlevel
                 then
11                 foreach Rule R_i in R do
12                     if component in R_i == C_i then
13                         if state in R_i ∈ sub-states of C_i at level =
                             currentlevel then
14                             truncate states with lower level
                                 sub-states;
15                             Rulesets[rulesetnum] ←
                                 Rulesets[rulesetnum] ∪ R_i;
16                     else
17                         if states in R_i ∈ any of states at level 0
                             then
18                             Rulesets[rulesetnum] ←
                                 Rulesets[rulesetnum] ∪ R_i;
19                     rulesetnum ← rulesetnum+1;
```

Algorithm 1 partitions the rules into rule sets. The algorithm generates rule sets until all levels of sub-states are covered. It initially generates rule set containing the higher level (abstract) representation of system. Once, this rule set is generated, each component that has sub-states is taken, and for each level of sub-states in component, treating states of other components to be at abstract level, rule sets are generated. Once all the rule sets are generated, each rule set is allocated to a thread in multithreading environment to perform rule expansion concurrently.

3.) Rule expansion: To analyze ONBs, we generate the system states from the component-states. We generate system states from component-states because, at any given instance, the state of system is dependent on all its components' states. In addition, we aim to find system states where components' states are conflicting or that might result in catastrophic behavior of system.

To simplify the system state generation process, we converted the rules into numerical form. The components are represented as rows and the states as columns and the ordinal positions of component and state are used to represent numerical values as shown in the upper table of Figure 2. For example, if there are *3* components in a system with *3* states each, then state 2 of component 3 is represented as (0, 0, 2) The sub-states

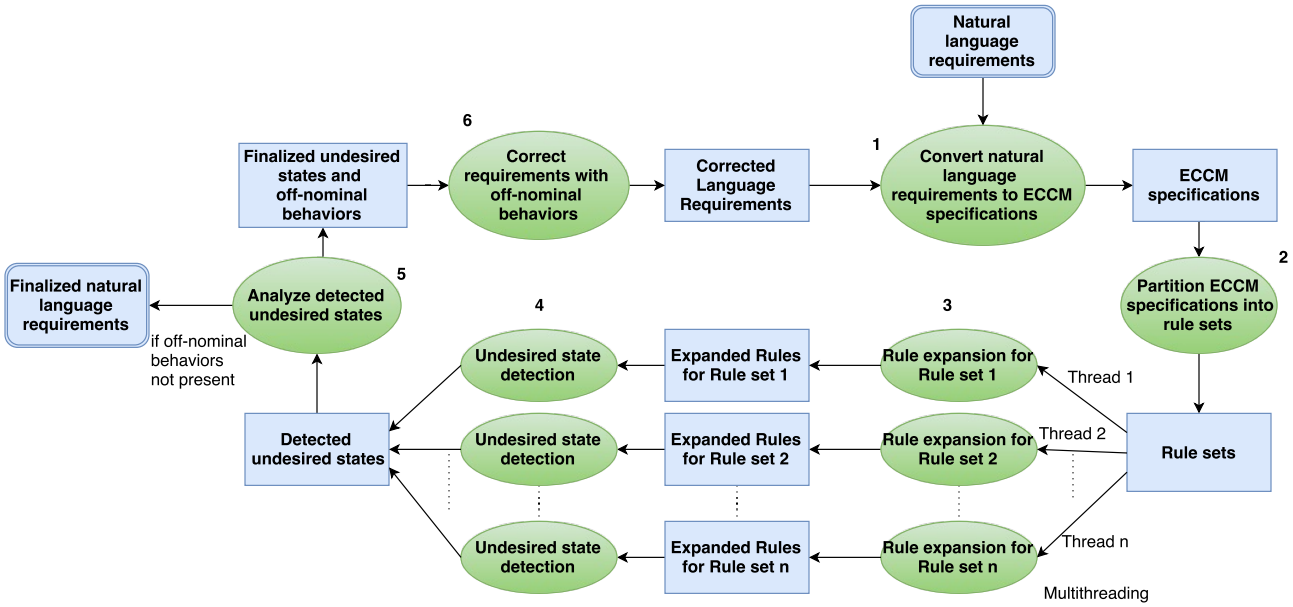

Fig. 1. Enhanced Causal Component Model Overview

are represented as sub-column within a column and the their position in the sub-column is appended to the parent state position with a separator '.' as shown in the sub-state table in Figure 2). For example, if state 1 of component 1 has 5 sub-states then we represent sub-state 2 of state 1 of component 1 as (1.2, 0, 0).

	State 1	State 2	.....	State n
Component 1	1, 0, 0	2, 0, 0	.....	n, 0, 0
Component 2	0, 1, 0	0, 2, 0	.....	0, n, 0
Component 3	0, 0, 1	0, 0, 2	.....	0, 0, n

Sub-State 1	Sub-state 2	.....	Sub-state k
0, 2.1, 0	0, 2.2, 0	.....	0, 2.k, 0

Fig. 2. Table for Numerical Representation of States and Sub-States of Components

We also converted any transition conditions with component-states into numerical form. These numerical states are used to generate numerical rules. Once the numerical rules are generated, the rules are expanded by performing absorption and propagation operations similar to CCM. We perform absorption and propagation to consider the component state information in transition condition and transfer it to the current and next system states. Absorption involves moving information from transition condition to the state if the transition condition is a system cause, system cause is the transition condition in which a component state or set of components' states result in state transition of other component); for instance, if there is rule like (0, 2, 0) : (2.1, 0, 0) $\rightarrow$ (2.4, 0, 0), it will be converted to (0, 2, 0) : (2.1, 2, 0) $\rightarrow$ (2.4, 0, 0). Propagation involves transferring the information from the current state to the next state. That is, after propagation, the above rule becomes (0, 2, 0) : (2.1, 2, 0) $\rightarrow$ (2.4, 2, 0). After the absorption and propagation, we consider the 0 values in the state, which means that the component-state can be any arbitrary state of that component. This process is referred to as an expansion and is performed to see all possible component state configurations possible for that given transition. As a result, we replace all the zeroes with possible component-states. We replace the system cause with a notation, for example, cause (0, 2, 0) is now denoted as T1. The rule (0, 2, 0) : (2.1, 2, 0) $\rightarrow$ (2.4, 2, 0) after expansion becomes T1: (2.1, 2, 2) $\rightarrow$ (2.4, 2, 2) and T1: (2.1, 2, 1) $\rightarrow$ (2.4, 2, 1).

4.) Undesired state detection: After the rules are expanded, the model is generated and a state profiling algorithm [7] is used to generate the number of incoming transitions due to environmental causes (EID), and due to system causes (TID), and the number of outgoing transitions due to environmental causes(EOD), and due to system causes(TOD) of a state in the model. The state profiling algorithm generates the above mentioned values by going through each state in the rule and creates or modifies corresponding profile. The algorithm identifies difference between system causes and environmental causes by looking over if the transition condition has any component state information. We perform this by checking the transition condition information with the database of components and states that we collected while creating the rules.

Based on the number of incoming transitions and the number of output transitions of each system state, we classified states as unrecoverable or unreachable.

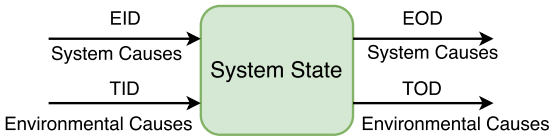

Fig. 3. State Profile with In-Degree and Out-Degree

Unrecoverable states are system states which cannot recover without external input, i.e, TOD = 0. These states can be undesired because the system is stuck in these states till external input is provided. Unreachable states are system states whose TID and EID are zero, which means that no system state can transition to this state. It is necessary to analyze such states for off-nominal behaviors, because while those states might be desired they can never be reached, leaving the system in a problematic state. However, all unrecoverable states might nor result in ONBs. Any unrecoverable state that does not lead to ONB is considered as false positive. These unrecoverable and unreachable states are used for finding ONBs.

5.) Analyze the detected undesired states: Not all unrecoverable/unreachable states are necessarily undesired states that result in ONBs. The requirements engineers and stakeholders should examine the detected off-nominal system states and label them as undesired or not. After the undesired states are finalized, their associated rules are examined to find ONBs.

6.) Correct requirements: The rules that result in ONBs and respective requirements are examined by requirements engineers and stakeholders, and corrective actions are taken based on stakeholders decisions. The entire analysis process is repeated with the corrected requirements until no undesired states are found.

IV. Case Study

To evaluate our proposed approach, we implemented the ECCM tool. The tool allows users to enter information about components, component states, component sub-states, and ECCM specifications (rules of component-state transitions). The tool utilizes a multi-threading environment to enhance performance. We performed case study using pacemaker using ECCM and compared it with CCM. The following subsections present the details of the pacemaker requirements, the procedure of the study, and the results.

A. Pacemaker Requirements

The pacemaker requirements document [12] is a 35-page document that describes the device's system requirements and provides details about diagnostics and therapy. The pacemaker system consists of a pulse generator (PG), a device controller monitor (DCM), leads, and a magnet. In this paper, we analyze the pulse generator, which plays a vital role in the system's operation; any malfunction of this component would result in complications for the patient.

B. Procedure of the Study

This section describes the procedure for our study.

TABLE I
Components and States

Component	States
Pulse Generator (PG)	off, on
Device Controller-Monitor (DCM)	off, on
Magnet	outofplace, inplace
Printer	off, on

1.) Convert natural language requirements to ECCM specifications: Two graduate students analyzed the pacemaker requirements document and identified the elements of ECCM: components, states, and transition conditions. Once the states were identified, hierarchical relationship among them is determined. Table I shows components and their higher level states that we identified. The pulse generator component has three levels of sub-states. The components, parent states, and sub-states are shown in Table II. The meaning of the acronyms of the sub-states of permanent, temporary, pace-now, magnet and power-on-reset states can be found in Table III. These sub-states represent the operating modes of the states of operation of the pacemaker.

Once the hierarchy of states is determined, the rules were created. In total, 71 rules are created, which serves as ECCM specifications.

2.) Partition into rule sets: In this step, we generated rule sets based on the sub-states generated in step 1. Using the rule set generation algorithm detailed in Section III, 4 rule sets are generated from 71 rules. Due to space limitations, we present a subset of rules from the first two rule sets as follows:

Rule Set 1 - First Two Rules:

1) $UserPress(on\_PG) : PG(off) \rightarrow PG(on)$
2) $UserPress(off\_PG) : PG(on) \rightarrow PG(off)$

Rule Set 2 - First Two Rules:

1) $Patient(chronic\_incompetence) :$
   $PG(on.preimplant) \rightarrow PG(on.implant)$
2) $Patient(sick\_sinus_{s}yndrome) :$
   $PG(on.preimplant) \rightarrow PG(on.implant)$

While the two rules in rule set 1 refer to user switching on and off the pulse generator (PG). The two rules in rule set 2 detail two conditions in which PG must be implanted into person. We can observe that all the rules illustrated above have environmental causes (Patient and UserPress are not components of system).

The first rule set has 7 rules out of 71 rules and the second rule set has 18 rules. The third and fourth rule sets contain 14 and 59 rules, respectively.

3.) Rule expansion: After the rule sets were generated, each rule set was allocated to a thread and the rule expansion is performed as explained in Section III to generate system states and their transition rules. We converted our rules into numerical rules, and the following are numerical rules created for the first two rules in rule set 2:

Rule Set 2 - First Two Numerical Rules:

1) $Patient(chronic\_incompetence) : 2.1, 0, 0, 0 \rightarrow 2.2, 0, 0, 0$
2) $Patient(sick\_sinus\_syndrome) : 2.1, 0, 0, 0 \rightarrow 2.2, 0, 0, 0$

TABLE II
COMPONENTS AND SUB-STATES

Component	Parent state	Parent state level	Sub-states
PG	on	0	pre-implant, implant, pre-discharge-follow-up, routine-follow-up, ambulatory, explant
PG	ambulatory	1	permanent, temporary, pace-now, magnet, power-on-reset
PG	permanent	2	off, DDDR, VDDR, DDIR, DOOR, VOOR, AOOR, VVIR, AAIR, DDD, VDD, DDI, DOO, VOO, AOO, VVI, AAI, VVT, AAT
PG	temporary	2	off, DDDR, VDDR, DDIR, DOOR, VOOR, AOOR, VVIR, AAIR, DDD, VDD, DDI, DOO, VOO, AOO, VVI, AAI, VVT, AAT, OOO, OAO, OVO, ODO
PG	pace-now	2	VVI
PG	magnet	2	off, DDD, DOO, AOO, VOO, OOO
PG	power-on-reset	2	VVI

TABLE III
EXPLANATION OF SUB-STATES WITH ACRONYMS IN PULSE GENERATOR

Category	Chambers paced	Chambers sensed	Response to sensing	Rate modulation
Meaning	O-None A-Atrium V-Ventricle D-Dual	O-None A-Atrium V-Ventricle D-Dual	O-None T-Triggered I-Inhibited D-Tracked	R-Rate modulation

Once the numerical rules are generated, absorption, propagation, and expansion are performed as explained in Section III. The result of these operations are the expanded rules that contain system states. The following are some of the expanded rules from the rule set 2:

Rule Set 2 - First Four Expanded Rules:

1) $Patient(chronic\_incompetence) : 2.1, 1, 1, 1 \rightarrow 2.2, 1, 1, 1$
2) $Patient(chronic\_incompetence) : 2.1, 1, 1, 2 \rightarrow 2.2, 1, 1, 2$
3) $Patient(chronic\_incompetence) : 2.1, 1, 2, 1 \rightarrow 2.2, 1, 2, 1$
4) $Patient(chronic\_incompetence) : 2.1, 1, 2, 2 \rightarrow 2.2, 1, 2, 2$

4.) Undesired state detection: The expanded rules from each rule set were analyzed to detect possible undesired states by generating state profile for each state as mentioned in step 4 of Section III. Using these state profiles, we found undesired states that aid in finding off-nominal behaviors (ONBs) using the process detailed in step 4 of Section III. The detailed results will be discussed in Section IV-C.

5.) Analyze detected undesired states: The detected undesired states were analyzed to eliminate false positives. False positives are the states which are desired but are identified as undesired. In our study, we found 146 false positives and they were easy to find due to the hierarchical relationship between super-state and sub-states. For example, states where PG is 'on' state or sub-states of 'on' state and DCM is 'on' state but the magnet is 'out of place' and the printer is 'off' are detected as undesired states. However, printer in 'off' state and magnet in 'out of place' state will not result in any off-nominal behaviors. Thus, these cases are be considered as false positives.

After eliminating the false positives, we obtained the finalized undesired states. We found 250 undesired states in the pacemaker requirements. For example, when the magnet state is 'in place' and pulse generator is not in 'magnet' state of operation, it is considered as an undesired state because when the magnet's state is in place, the system requires the operating mode node not to have any sensing operation. Having the sensing operation when the magnet's state is in place might result in energy interference, which can cause the electrical problem with the pacemaker.

6.) Correct requirements: Once the undesired states are identified, their corresponding rules and requirements are identified. In our study, we manually traced back from rules to requirements. Consider a case that shows an undesired state, 2.5.4.4, 2, 1, 2, which indicates magnet is out of place but PG is in magnet state. Its corresponding rule is $User(remove\_magnet) : 2.5.4.4, 2, 2, 2 \rightarrow 2.5.4.4, 2, 1, 2$ and its requirement is "*When the magnet is removed the device shall assume pretest operation*". Because the pretest operation is never defined in the document, it is hard to know what transition occurs from magnet state and what the destination state is. To clarify this requirement, we corrected it as follows: *When the magnet is removed, the device shall assume pretest operation and comes out of the magnet state*.

We also found ONBs from the states that are associated with PG and DCM. It is not safe for DCM to be switched off when PG is on if there is a necessity for continuous monitoring. When continuous monitoring is required, this situation will result in complications of patient's condition because the change of pacing and sensing functions can no longer be controlled and it will affect the heart beat rate of the patient. If such incomplete requirements are not addressed, it is likely to produce a product that can threat the patients' safety.

C. Results

Table IV shows the results of our study. RS in the table refer to the rule set. Because CCM does not use the hierarchical approach, some items are not applicable such as the number of sub-states and rule sets. We explain the results obtained from the ECCM first and then compare them with those from the CCM.

As Table IV shows, using ECCM approach, we identified 4 components, 8 states (level 0 states or parent states or super states), 61 sub-states for 3 levels of hierarchy. 71 rules were created and 4 rule sets were obtained. The total number of system states generated is 504, and the number of expanded rules for all rule sets is 1880. The ECCM approach detected

TABLE IV
RESULTS OF ECCM AND CCM

	ECCM	CCM
Number of components	4	4
Total number of states	8	62
Total number of sub-states	61	N/A
Number of rules from NL requirements	71	1192
Number of rule sets generated	4	N/A
Number of system states analyzed	16 (RS1), 48 (RS2), 40 (RS3), 400 (RS4)	448
Number of expanded rules generated and analyzed	56 (RS1), 232 (RS2), 160 (RS3), 1432 (RS4)	9496
Number of state analyzed for finding undesired states	396	448
Number of ONBs	3	3
Time taken to convert NL to rules (after reading requirements)	1 hour	4 hours
Time taken for analysis of false positives	30 minutes	0
Time taken for generation of undesired rules	0	4 hours

396 undesired (off-nominal) states and 146 of them were false positives. We found 3 ONBs from 250 undesired states. The identified ONBs are detailed in Table V. Every undesired state is caused by one of those 3 off-nominal behaviors. We consider the detected ONBs as low to moderate risk as they can be easily addressed, and are not likely to lead to fatal accidents unless some additional factors come into play such as energy inference.

When we applied the CCM approach to the same set of requirements, because the process of identifying components is identical to ECCM, we obtained the same number of components, which is four. However, the number of rules created for CCM (1192) is far larger than for ECCM (71). Also, the number of expanded rules bt CCM (9496) is much larger than ECCM (1880). Unlike ECCM, CCM does not detect possible undesired states but requires human intervention to analyze all states to find undesired states, which in turn are used for generating undesired rules to check the reachability of the undesired states. Thus, it does not produce false positives. By analyzing 52 undesired rules, we were able to detect 3 ONBs that were found from ECCM. We are able to find two ONBs at level 0 of hierarchy and one ONB at level 2 while the total number of levels are 3. While the two approaches found the same number of ONBs, the systematic way of the ECCM approach located them more easily and faster. Further, CCM does not guarantee to locate ONBs.

Also, we found that our approach is more efficient than the CCM approach. As shown in Table IV, the ECCM approach took 1 hour to convert natural language requirements into rules while the CCM approach took 4 hours. This indicates that creating rules using ECCM is much easier than CCM. Analyzing ONB is done differently for each approach. While the analysis of ONBs in ECCM involves automated possible undesired state detection followed by false positive analysis, the analysis of ONBs in CCM involves manual identification of undesired rules. These undesired rules are used to detect possible ONBs in expanded rules automatically. Thus, for the ONB analysis time, ECCM took 30 minutes, but CCM took 4 hours (due to large number of rules). Because we used only a small set of requirements (pulse generator) in this study, if we apply our approach to a larger set of requirements, ECCM will produce far greater benefits than CCM.

TABLE V
LIST OF OFF-NOMINAL BEHAVIORS (ONBS) FOUND

S.No	Off-nominal behavior
1	Magnet is inplace when PG is off
2	DCM is switched off when Pulse Generator is on
3	Pulse generator is in magnet state when magnet is out of place or Pulse generator is in other state when magnet is in place

V. DISCUSSION

By examining the results, we drew the following observations. First, overall, the proposed technique helps to determine ONBs in systems with complex behaviors during the requirements phase. Second, the proposed approach reduces the number of rules and states being analyzed, as we divide the input rules into rule sets. The results indicate that state and rule explosion can be reduced by combining state abstraction and state space restriction.

We believe that our hierarchical approach can help the requirements engineers and stakeholders to understand the dependencies among the states more easily and address of ONBs compared to non-hierarchical approaches such as CCM. In addition, our proposed approach can give system behavior at abstract level which we believe make it comprehensible for non-technical stakeholders as abstract level representation is simple.

Although our results are promising, our approach has some limitations that we want to address. One limitation is that our approach cannot guarantee exposure of all ONBs as it does not consider states that recover to another undesired state. We plan to address this limitation by analyzing all possible states and then profiling rules similar to CCM.

The second limitation of our approach is that it is still prone to state explosion problem. If there are too many states at a single level of hierarchy, the approach can have scope for state explosion. We plan to mitigate it by considering combinatorial hierarchical exploration. The third limitation is the high number of false positives during automated undesired state detection. We plan to address it by creating a knowledge base which aids in identifying domain specific off-nominal properties.

VI. RELATED WORK

To date, some amount of research has been done in finding ONBs [3], [2]. For example, Sukhwani et al. [6] proposed software reliability growth models to find ONBs during the

maintenance phase. Other researchers have tried to analyze ONBs at the testing phase. Foyle and Hooey [5] proposed off-nominal testing by incorporating off-nominal scenarios into experimental design.

Some researchers have analyzed ONBs by developing simulation models [4]. A few researchers have tried to detect the ONBs at the requirements level. For example, Tsai et al. [13] analyzed scenario specifications for off-nominal scenarios using automated event tree analysis, in which the scenarios are executed by considering failure models to generate event trees. Day et al. [2] used system modeling language(SysML) to model generic patterns of ONBs and used them to check the completeness of the system.

These methods consider off-nominal situations from human factor point of view. Unlike these methods, in our work, we consider exposing the causes that result in ONBs from system's perspective.

VII. Conclusions and Future Work

In this paper, we presented the model-driven requirements analysis approach that reduces state and rule explosion problems using a hierarchical model exploration strategy, and its empirical results with the pacemaker requirements.

Although our empirical results showed the promising results, our approach also has several limitations as we discussed in Section V. For future work, we intend to investigate and improve our approach further considering those limitations. First, we plan to use natural language processing to ease process of identifying model elements, add analysis of orthogonal states to proposed ECCM model, and reduce false positives using a knowledge base for identifying undesired states. Second, we plan to perform various empirical studies to explore further advantages and limitations of our approach. Third, we also plan to adapt and use the rule set generation algorithm for system of systems. Finally, we plan to take possible contexts into consideration when analyzing and exposing off-nominal behaviors.

Acknowledgment

This work was supported, in part, by NSF CAREER Award CCF-1564238 to University of North Texas.

References

[1] A. K. Mostafa, "How embedded systems benefit people and in what fields ?" Tech. Rep. July, 2015.

[2] J. Day, K. Donahue, A. Kadesch, A. Kennedy, and E. Post, "Modeling off-nominal behavior in SysML," in *AIAA Infotech*, Jun. 2012, pp. 19–21.

[3] D. Firesmith, "The Need to Specify Requirements for Off-Nominal Behaviors," 2012. [Online]. Available: https://insights.sei.cmu.edu/sei_blog/2012/01/the-need-to-specify-requirements-for-off-nominal-behavior.html

[4] G. Fraccone, V. Volovoi, A. Colon, and M. Blake, "Novel air traffic procedures: Investigation of off-nominal scenarios and potential hazards," *Journal of Aircraft*, vol. 48, no. 1, pp. 127–140, 2011.

[5] D. Foyle and B. Hooey, "Improving evaluation and system design through the use of off-nominal testing: A methodology for scenario development," in *Internaltional Symposium on Aviation Psychology*, 2003, pp. 397–402.

[6] H. Sukhwani, J. Alonso, K. S. Trivedi, and I. Mcginnis, "Software reliability analysis of nasa space flight software: A practical experience," in *2016 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, Aug 2016, pp. 386–397.

[7] D. Aceituna and H. Do, "Exposing the susceptibility of off-nominal behaviors in reactive system requirements," in *Requirements Engineering*, Aug. 2015, pp. 136–145.

[8] B. Boehm and V. Basili, "Software defect reduction top 10 list," *IEEE Computer*, pp. 135–137, 2001.

[9] J. M. Stecklein, J. Dabney, B. Dick, B. Haskins, R. Lovell, and G. Moroney, "Error Cost Escalation Through the Project Life Cycle," *Incose -Annual Conference Symposium Proceedings*, 2004.

[10] X. Devroey, G. Perrouin, M. Cordy, A. Legay, P. Schobbens, and P. Heymans, "State machine flattening: Mapping study and assessment," *CoRR*, 2014.

[11] A. Valmari, *The state explosion problem*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 429–528.

[12] Boston Scientific, "Pacemaker system specification," Boston Scientific, Tech. Rep., 2007.

[13] W. Tsai, W. Song, R. Paul, Z. Cao, and H. Huang, "Services-oriented dynamic reconfiguration framework for dependable distributed computing," in *COMPSAC*, vol. 1. Citeseer, 2004, pp. 554–559.