2019 IEEE/ACM 2nd International Workshop on Robotics Software Engineering (RoSE)

Exposing Off-Nominal Behaviors in Multi-Robot Coordination

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*—Often software in robotics systems is susceptible to unexpected and unforeseen behaviors called off-nominal behaviors (ONBs) and these ONBs can affect the reliability or safety of the systems. While some work is done on exposing ONBs in a system, there has been little research conducted on exposing ONBs when multiple robots perform a task together. In this paper, we propose a combinatorial based approach to expose ONBs in such multi-robot coordination tasks during the requirements engineering phase. Our approach separates system level analysis and coordination level analysis, and generates combinations that need to be manually analyzed for ONBs. To evaluate the effectiveness of our approach, we conducted an empirical study with a set of requirements that have three coordination tasks. The results of our study show that our approach offers a means for ONB knowledge acquisition and reduces significant human effort and time required for exposing ONBs.**

I. Introduction

When engineers verify robotics systems, they tend to focus on robots' expected behaviors, but ignoring unexpected behaviors can result in catastrophic situations such as significant financial losses or human losses [1]. The behaviors that are unexpected and cannot be foreseen are called off-nominal behaviors (ONBs) [2]. An example of ONBs is a robot not stopping when its ultrasonic sensors detected an object ahead. Because the consequences of not addressing ONBs can be hazardous and even be life-threatening, ONB analysis has been applied for safety-critical systems including robotics [3]. Although ONB analysis is done mostly after implementation or during testing [4], some researchers have proposed techniques that apply ONB analysis during the requirements engineering phase [5] or early design phase [2]. For example, Combi-CCM [5] identifies ONBs by analyzing requirements and corrects requirements that cause ONBs.

While these approaches expose ONBs in robotic systems and aid in making the robot more robust and reliable, none of them performed ONB analysis considering robot coordinations. In the robotics domain, often multiple robots need to collaborate to make a decision or to perform a task that cannot be done by a single robot alone (e.g., moving heavy objects and path planning in a complex environment with many obstacles). When multiple robots collaborate together to perform certain tasks, we refer to such tasks as coordination tasks, and this type of collaboration is referred to as multiple robot coordination (e.g., two robots moving an object together) [6]. Analyzing ONBs considering coordination in multi-robot systems is important and necessary because the damage from system failures would be magnified to a great extent as multiple systems and external objects are involved in the coordination task, and in environments where systems perform tasks in presence of humans or have human interactions, ONBs in coordination tasks can result in casualties (e.g., a heavy object being carried by large industrial robots falling on a person).

Some researchers have attempted to model and analyze multiple robot coordination in the early design phase [6], [7], [8]. For example, Matson and DeLoach proposed an organization-based multi-agent system architecture (OMAS) model [7], which makes a robot reliable by reassigning robots new tasks when there is a sensor failure or malfunction in one of the systems performing a task. While these approaches make coordination tasks reliable by providing flexible ways to achieve the given coordination tasks, none of them considers ONBs, which can be great threats for such systems.

To address the limitations of existing approaches including Combi-CCM, we propose a combinatorial approach to analyze ONBs in multi-robot coordination. Because our proposed approach requires modeling the behaviors of robots, we used Combi-CCM to obtain model elements. Using these model elements, we performed the coordination task analysis to expose ONBs at both system and coordination levels by analyzing the combinations of states generated using IPOG algorithm [9]. To evaluate our approach, we performed an empirical study using three multi-robot coordination tasks. The results show that our approach reduces analysis time and effort and finds ONBs that can help in reducing the defects that can propagate to later phases of software development.

II. Background

This section describes model elements and Combi-CCM approach that are used as part of our proposed approach.

A. Model elements

To perform ONB analysis, we define the model elements at two levels: a system level and a coordination task level. The model elements at the system level are components, their states, system states, and transition conditions (similar to Combi-CCM [5]). The model elements at the coordination task level are coordination entities, entity states, and coordination states.

1) Model elements for the system level ONB analysis: A system comprises a set of components. For example, in a simple robotic system, a set of components would be {*switch, motor*

978-1-7281-2249-6/19/$31.00 ©2019 IEEE
DOI 10.1109/RoSE.2019.00006

controller, ultrasonic sensor, light sensor}. Every component has different states that are defined based on its functionality. For example, a set of component states for the component *motor controller* is {*off, on.idle, on.move, on.rotate*}. The notation 'a.b' indicates 'b' is a sub-state of state 'a'. For example, in the set of states of a *motor controller*, 'on.move' indicates 'move' is a sub-state of state 'on' of a *motor controller*. A system state is defined as combination of concurrent component states of all components in a system at a given time. A system can have many system states. An example of a system state is {*switch(on), motor controller(on.move), ultrasonic sensor(on.no detect), light sensor(on.detect)*}

A transition condition is a trigger or an event that can result in a change of a component's state. A transition condition can be a change in the operating environment such as a change in room temperature, or can be human interactions with a system, which can result in a state transition of a component. We refer to such transition conditions as environmental transition conditions. A transition condition can also occur due to a change of states of other components in a system. We identify such transition conditions as system transition conditions. An example of an environmental transition condition is *the presence of an obstacle that can change the state of an ultrasonic sensor*, and an example of a system transition condition is the component state 'Switch(on)' as it results in state changes of a motor controller in the system.

Using the identified components, their states, and transition conditions, we create transition rules, which are in the form of *Transition Condition : Component(Current state)* $\rightarrow$ *Component(Next state)*. Every system has a set of transition rules and they are used in assessing the causes of ONB problems.

2) Model elements for the coordination level ONB analysis: A coordination task involves task specific features (e.g., the type of tasks, the roles of systems in performing a task, and the objects other than systems involved in performing a task) that need to be considered in addition to the behaviors of the robots. To take such coordination task specific behaviors into account, we identify coordination entities. Coordination entities include components of systems involved in the coordination task that directly affect the coordination task.

For example, consider two simple robotic systems performing a coordination task of moving a block placed on top of them from location A to location B. As mentioned earlier, each robotic system has 4 components. Among them, only the motor controller component of the robotic systems can affect the coordination task, i.e., a movement of the block placed on robots to a destination location. We consider these motor controllers as part of coordination entities. We denote them as *System name.component name*. Other entities that are part of coordination entities are objects and their features (e.g., an object's location) that form the coordination task itself. For example, in the above example, a coordination task is to move a block placed on top of two robots from a start location to a destination location. The block itself and the location of the block determine the completion of the coordination task. First, we need to have a block on the robots for them to move, and the block should not be on the ground. Second, the robots carrying a block should not stop in the midway. To analyze this behavior, we consider a block, and a block location also as part of coordination entities. Therefore, each coordination task has a set of coordination entities. From our example, we have coordination entities as follows: {Robot1: motor controller, Robot2: motor controller, block, block location}.

Similar to components, coordination entities also have states. For example, the states of coordination entity *block* are: 'on robots', 'not on robots'. If any of the coordination entity's states are continuous, we convert them to discrete states. For example, the coordination entity *block location* usually has a series of continuous latitudinal and longitudinal values based on the fixed axis. We convert them into discrete values and generate states like 'start', 'on the way', and 'destination'. Thus, the states of *block location* are {start, on the way, destination}.

To assess the state of a coordination task at a given time, we combine the concurrent states of all coordination entities to form a coordination state. In our example, one of the coordination states is {Robot1:motor controller(on.move),Robot2:motor controller(on.move), block(on robots), block location(start)}

We use system states and coordination states to perform ONB analysis at a system level and a coordination level, respectively.

B. Combi-CCM

Combinatorial Causal Component Model (Combi-CCM) proposed by the authors of this paper [5] reduces the manual effort required for ONB analysis of a system (from years and days to hours). In this paper, we incorporate Combi-CCM to analyze ONBs at the system level as well as coordination level. Our goal is to identify possible ONBs in multi-robot coordinations with less human effort and time. We briefly summarize the steps of applying Combi-CCM (the details can be found in [5]). From the given set of natural language requirements, we identify the model elements and write component state transition rules (as described in Section II-A1). We convert these component state transition rules to system state transition rules by a series of operations such as expansion, absorption, and propagation [5]. To reduce the number of states to examine, Combi-CCM generates 'n'-way combinations of component states using IPOG [9] ('n' is determined based on causal dependencies among components). These combinations are manually analyzed to find undesired system states. The undesired system states are used on system state transition rules to find the occurrence of ONB and the recoverability of an ONB is analyzed using a state profiling algorithm. The causes of ONBs and recoverability are discussed among stakeholders and domain experts, and requirements are corrected.

III. The Proposed Approach: CorcoCCM

Figure 1 illustrates our approach Coordination-Combi-CCM (CorCoCCM). The ovals represent the processes, the document symbol represents input requirements, and the rectangles represent inputs and outputs of the processes. The numbers over ovals represent the step numbers. Our approach provides a means to manually analyze ONBs that can occur during the multi-robot coordination tasks with less effort. The detailed descriptions of each step in CorCoCCM are as follows.

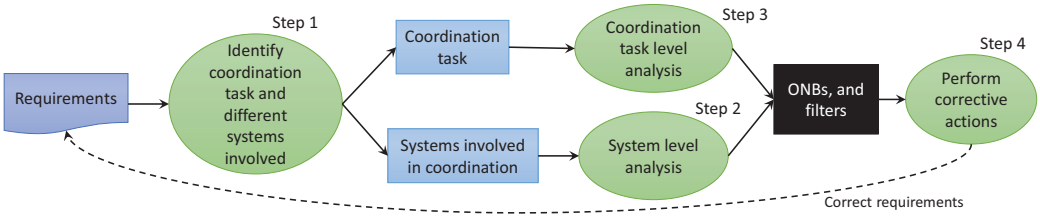

Fig. 1. Overview of proposed methodology

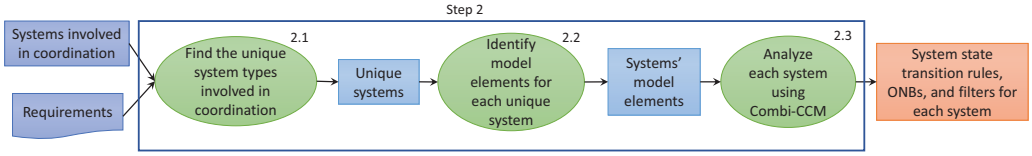

Fig. 2. Approach for performing ONB analysis on systems participating in multi-robot coordination task

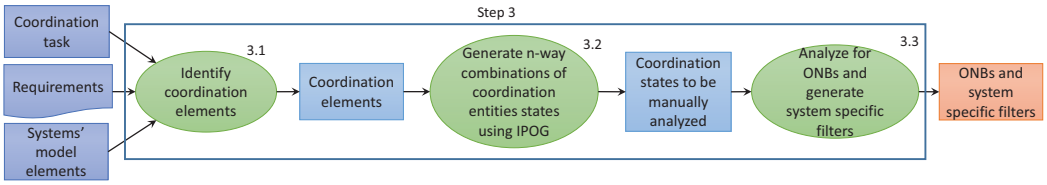

Fig. 3. Approach for performing ONB analysis on coordination aspect of multi-robot coordination task

(Step 1) Identify coordination tasks and their associated systems: The first step of CorCoCCM is for stakeholders to identify the coordination tasks and robotic systems that participate in the tasks. We perform this step to expose ONBs that can affect coordination tasks and to ensure that the systems are reliable at both system and coordination levels. It is possible that similar types of systems can have different roles when performing a coordination task. We identify this information as it helps in system and coordination level analyses. Consider the example described in Section II. An object needs to be moved from position A to position B. A team of two robots is employed to move the object. Every robot has an ultrasonic sensor, motor controller, light sensor, and switch. The ultrasonic sensor identifies obstacles and a distance to the destination while the light sensor identifies the destination as well as other robots within the team. The switch activates the robot and the motor controller controls the movement of the robot.

(Step 2) System level analysis: Once we identify systems and coordination tasks, we perform the system level analysis to check whether each system is reliable. To do so, the following sub-tasks are performed (Figure 2).

Task 2.1: Find the unique system types involved in coordination: The first step in the system level analysis is to find system types, along with their roles. If two robotic systems have the same hardware and perform similar functionalities, we consider that these two systems have the same system type. However, if the two systems have different hardware and perform different functionalities, they have different system types. It is possible for two systems to have similar hardware and functionalities but to perform different roles in a coordination task. We consider such systems to have different roles. We find unique system types to reduce the redundant analysis. If two systems belong to the same type, analyzing the system type as a whole implies analyzing both the systems. The roles, on the other hand, help in identifying coordination elements (explained in the later sections). In the prior example, the two robots involved in the coordination task are similar, and they play the same role. Hence, we consider we have only one system type. However, if one of the robots has a different set of sensors, we will be having two system types. In our example, both robots play a similar role, i.e., acting as a transporter of an object. However, if one of the robots observes surroundings and other robot moves based on those observations, we consider the robots to have different roles.

Task 2.2: Identify model elements for each unique system: Once the system types are found, we identify model elements for each system. Finding model elements from the requirements can be challenging because often the requirements are incomplete and their description style varies widely. In such cases, we start by identifying the components of the system. We then identify/define possible states of the components. We also identify transition conditions, which can result in a state change in a component. In our example, we have a single system type, and its components are {switch, motor controller, ultrasonic sensor, light sensor}. The states of these components are as follows: {switch : {off, on}, motor controller: {off, on.idle, on.move, on.rotate}, ultrasonic sensor: {off, on.nodetect, on.detect}, light sensor: {off, on.nodetect, on.detect}}. An example of transition conditions is user pressing the button, which causes the switch to transition from 'off' to 'on'.

Task 2.3: Analyze each system using Combi-CCM: Once all model elements are identified, we write transition rules in the form of: *Transition Condition: Component(current state) → Component(next state)*. These rules and model elements are processed by Combi-CCM [5] to generate system state transitions. Combi-CCM generates combinations of states using IPOG algorithm [9]. These combinations are manually analyzed to find ONBs, and filters are generated. The outcome of the system level analysis is a set of system state transition rules, a list of ONBs, and filters. The filters are numerical expressions, which are used to identify system states that have ONBs. In our example, a possible ONB is a motor controller moving when an ultrasonic sensor detected an object. The filter we generate will be in the form of $x_1.x_2 \cdots x_n$, where x_i is the state number of i^{th} component, and 'n' is total number of components in the system. If an x_i value is 0, it means it is a 'don't care' and the component can be in any of its state. For the ONB in our example, the filter would be 0,3,3,0.

(Step 3) Coordination level analysis:: Once the system model elements and system roles are determined, we perform a coordination level analysis through the following sub-tasks.

Task 3.1: Identify coordination elements: As explained in Section II, coordination elements are coordination entities and their states. To find coordination entities, we check what entities other than the system under consideration are involved in the task and consider them as part of coordination elements. We identify coordination entities rather than considering all components from all systems because we intend to reduce the complexity and human effort required for the ONB analysis. In our example, the coordination elements are the object placed on the robots, its location, motor controllers in robots 1 and 2. We denote the components found in the system using the following notation: system-name.component-name (e.g., Robot1.motor controller). Having found the coordination entities, we identify their states. The states of the components in the system will be the same as their states found during the system level analysis. We identify states of other coordination entities that are not part of the system based on the coordination task. For example, for the object being carried, its states can be 'on the robots' and 'not on the robots', and for the object location, the states can be 'start', 'on the way', and 'destination.'

Task 3.2: Generate n-way combinations of coordination entities states using IPOG: To analyze ONBs at the coordination level, we consider the behaviors of the systems involved in the task as well as the task itself. One way to find problems with coordination is to consider all possible combinations of all systems' states and task-specific states. The problem with such an approach is that it can result in a large number of states. To avoid this problem, we identify coordination entities and their states, and apply IPOG algorithm [9]. The IPOG algorithm generates n-way combinations of coordination entity states, which produce concurrent states of all coordination entities.

The 'n' value is determined based on coordination dependencies, i.e., dependencies among coordination entities, where a change in a state of one entity can result in changes in states of other entities. System and coordination levels can have different 'n' values. For example, a system level analysis may require 3-way combinations whereas a coordination level analysis may only require 2-way combinations because the causal dependencies among components and among coordination entities can be different. In our example, only the motor controller component affects the coordination task. Thus, the 'n' value will be a 2-way as we need to consider possible behaviors of a motor controller performing a task. However, if there are two motor controllers in a system which perform a task, and they have a dependency relationship, then we use a 3-way. In our example, we generated a total of 16 coordination states and one of them is: {Robot1.motor controller(on.rotate), Robot2.motor controller(on.move), Object(on robots), Object location(start)}.

Task 3.3: Analyze for ONBs and generate system specific filters: The coordination states generated using IPOG are manually analyzed to find ONBs. In our example, some of the ONBs we found are: robots moving without the object on them, object being dropped at a place that is not the destination. To address these ONBs, we find their causes and the context that lead to ONBs. We can identify the causes of ONBs by building system specific filters. We build filters separately for each system based on the states of the components in these systems that result in ONBs at the coordination level. If there is only one component in a system among coordination entities, we can forgo the step of generating filters and analyze the ONB causes by searching in component state transition rules. In our example, we have only one component among coordination entities for each system, therefore we find causes from the component state transition rules of the system defined during the system level analysis. The additional knowledge about situations in which the coordination state can be an ONB is collected through discussion among stakeholders and domain experts. Once the causes of the ONBs are found, the requirements will be corrected by stakeholders and experts.

IV. Empirical Study

To evaluate our approach, we conducted an empirical study considering the following research questions:

RQ1: Does CorCoCCM reduce human effort and time required to expose ONBs in multi-robot coordination scenarios?

RQ2: Does CorCoCCM maintain its ONB exposure ability despite decreasing human effort and time required for analysis?

A. Objects of Analysis

We considered three coordination tasks, which can be performed by small robots such as s-bots, foot-bots, and eyebots. S-bots are simple insect like robots developed as part of swarm-bots project [10] supported by the European commission. Foot-bots and eye-bots are part of swarmanoid project [11] done by various European institutes. Foot-bots are similar to s-bots and are used for moving through rough terrains. Eye-bots, on the other hand, are able to fly and are used to sense and analyze the environment.

Table I shows the descriptions of the coordination tasks that we consider in this study. We used these tasks to analyze how the problems found in these projects can be found earlier using

our approach. These coordination tasks contain tasks that are performed by the same type of systems and different types of systems. While two tasks, RCO and RPB, are performed using the same type of robotic systems, RGR is performed using different types of robotic systems. While the detailed requirements of these robotic systems can be found in [10], [11], we extracted a subset of requirements that describe these coordination tasks from [12], [13], [14].

TABLE I
COORDINATION TASKS

ID	Reqs. Name	Description
1	Robots carrying an object (RCO)	Two s-bots [10] try to move an object from location A to location B together.
2	Robots placing object over block (RPB)	Two s-bots need to place an object over a block which is higher and requires one s-bot to lift other s-bot [10] to place object on the top of the block.
3	Robots guiding other robot (RGR)	Two eyebots [11] help a footbot [11] to reach destination.

B. Variables

1) Independent Variables: The independent variable is the ONB analysis technique. We consider two controls that do not consider roles and types of systems and one heuristic as follows: (Note that we selected control techniques that use the combinatorial approach to have fair comparisons.)

- Control I (Combi-CCM for the entire systems and coordination tasks (*CoSysCo*)): This technique combines all systems and coordination entities together and generates n-way combinations of states. *CoSysCo* does not perform separate system level and coordination level analyses.
- Control II (Cartesian-Combi-CCM (CaCoCCM)): This technique applies Combi-CCM for each system and coordination task-specific entities individually. It then performs a Cartesian product of the states to generate combinations of states for ONB analysis.
- Heuristic (Coordination-Combi-CCM (*CorCoCCM*)): The heuristic technique separates system level and coordination level analyses while generating combinations of states, and considers only coordination entities for generating combinations for the coordination level analysis.

2) Dependent variables: For RQ1, we used three dependent variables: 1) the number of combinations at the system and coordination level analyses; 2) the size of combinations at the system and coordination levels (i.e., the number of components or coordination entities in combinations required to be manually analyzed); and 3) the time taken for ONB analysis. For RQ2, the dependent variable is the number of ONBs found.

C. Experimental Procedure

To perform our experiments, we used the Combi-CCM tool [5] implemented in Java, and ACTS tool [15], a combinatorial testing tool that uses the IPOG algorithm [9]. Combi-CCM uses ACTS internally to generate combinations of states.

To collect data for our proposed approach (CorCoCCM), we followed the steps detailed in Section III. One graduate student with the help of the domain experts identified the model elements of systems, roles of systems, types of systems and coordination entities from the descriptions of coordination tasks manually. We then generated transition rules and analyzed coordination entities to find the 'n' value. In our study, we found 3-way to be reasonable for generating combinations of states for the system and coordination level analyses because for all three tasks, at most three coordination entities are causally dependent on each other at any given point of time. We also considered 2-way combinations because we wanted to check if 2-way can identify the same ONBs with a smaller number of states when compared to 3-way.

To collect data for two control techniques, we used the same set of model elements and rules identified from the requirements of 3 tasks for our heuristic approach. Similar to our heuristic approach, we also generated 2-way and 3-way combinations for CoSysCo and CaCoCCM for ONB analysis. In CoSysCo, we grouped all components and coordination entities to generate combinations whereas in CaCoCCM, we performed the individual system analysis along with external objects involved in the coordination task similar to the analysis performed in CorCoCCM (but without considering their roles or types), and then performed a Cartesian product to generate combinations. In the case of CoSysCo, there is no separate set of states for the system level and coordination level analyses. We analyzed the same set of combinations for ONB analysis at both levels. After collecting all the data, we measured the number of combinations, size of combinations generated by each technique at the system and coordination levels. We also measured the time taken for ONB analysis as well as the number of ONBs found to evaluate the efficiency and effectiveness of each technique. For the techniques that produced more than 1000 combinations, we randomly selected 400 combinations, using which we estimated the results.

D. Empirical Results

1) RQ1 analysis: RQ1 investigates whether our proposed approach can reduce the human effort and time to expose ONBs under multi-robot coordination scenarios. To answer this question, we measured the number of combinations to be manually analyzed, the size of each combination (i.e., the number of components and coordination objects or coordination entities whose states are included in the combination), and the time taken for analysis. Because we analyzed ONBs at the system and coordination levels for our heuristic technique, we illustrate the results for both system and coordination level analyses in Table II and Table III, respectively.

As Table II shows, overall, CorCoCCM produced smaller numbers of combinations with smaller or similar combination sizes for both 2-way and 3-way when compared to control techniques. In the table, RGR lists two sets of numbers under 'Size of combinations' for CaCoCCM and CorCoCCM because RGR has two different types of systems (eyebot and footbot). The numbers in parentheses represent the number of combinations of states with the corresponding size of combinations. For example, in CorCoCCM, it lists 11 (15) and 14 (21). This means there are two different sizes of combinations: 11 and 14, and there are 15 combinations with size 11 and 21 with size 14.

TABLE II
NUMBER OF COMBINATIONS AND THEIR SIZE MANUALLY ANALYZED FOR SYSTEM LEVEL ANALYSIS

Req.	CoSysCo				CaCoCCM				CorCoCCM			
	2-way		3-way		2-way		3-way		2-way		3-way	
	No. of comb.	Size of comb.	No. of comb.	Size of comb.	No. of comb.	Size of comb.	No. of comb.	Size of comb.	No. of comb.	Size of comb.	No. of comb.	Size of comb.
RCO	46	70	303	70	70	34	410	34	35	34	205	34
RPB	46	70	303	70	70	34	410	34	35	34	205	34
RGR	32	42	189	42	51	11 (30), 14 (21)	235	11 (118), 14 (117)	36	11(15), 14 (21)	176	11 (59), 14 (117)

TABLE III
NUMBER OF COMBINATIONS AND THEIR SIZE MANUALLY ANALYZED FOR COORDINATION LEVEL ANALYSIS

Req.	CoSysCo				CaCoCCM				CorCoCCM			
	2-way		3-way		2-way		3-way		2-way		3-way	
	No. of comb.	Size of comb.	No. of comb.	Size of comb.	No. of comb.	Size of comb.	No. of comb.	Size of comb.	No. of comb.	Size of comb.	No. of comb.	Size of comb.
RCO	46	70	303	70	7350	70	252150	70	41	28	262	28
RPB	46	70	303	70	7350	70	252150	70	41	32	264	32
RGR	32	42	189	42	37800	42	9774648	42	30	17	163	17

TABLE IV
TOTAL TIME TAKEN FOR ONB ANALYSIS AT SYSTEM LEVEL AND COORDINATION LEVEL (IN HOURS)

Req.	CoSysCo		CaCoCCM		CorCoCCM	
	2-way	3-way	2-way	3-way	2-way	3-way
RCO	2.67	5.25	61.44	2102.4	**1.30**	4.30
RPB	2.78	5.35	61.68	2103.98	**2.05**	4.45
RGR	1.45	3.75	160	41374	**1.05**	3

For 2-way combinations, CorCoCCM produced 35 and 35 in RCO and RPB, which are smaller than those produced by control techniques. However, in the case of RGR, CoSysCo produces a smaller number of combinations (32) when compared to CaCoCCM (51) and CorCoCCM (36). Similarly, for 3-way, CorCoCCM produced better results compared to the control techniques for all three tasks. While it produced 205, 205, and 176 combinations for RCO, RPB, and RGR, respectively, CoSysCo produced 303, 303, and 189 and CaCoCCM produced 410, 410, 235. By comparing the numbers of combinations produced by 2-way and 3-way, we can see that the numbers produced by 3-way are much larger than those produced by 2-way. When we examined the size of combinations, we can see that CoSysCo produced the larger sizes of combinations compared to those produced by CorCoCCM and CaCoCCM for all tasks for both 2-way and 3-way.

Similar to the results at the system level analysis, the results at the coordination level (Table III) show that our approach, CorCoCCM, produced the smallest numbers of combinations with the smallest sizes for all three tasks. CaCoCCM produced the largest numbers of combinations of states, which are much larger compared to two other techniques. For example, for 3-way combinations in RGR, CaCoCCM produced 9774648 combinations of states making manual analysis almost impossible whereas CorCoCCM and CoSysCo produced 163 and 189 combinations, respectively, reducing the human effort to a great extent. The reason why CaCoCCM produced such large numbers is that CaCoCCM produced a Cartesian product of combinations of states generated at the system level along with the combinations of objects that are part of a coordination task and their properties. For example, CaCoCCM produced 410 combinations (3-way) for RCO, which contains 205 combinations from each robot system. The number of combinations of objects that are part of a coordination task and their properties is 6. Therefore, upon performing a Cartesian product, we get 252150 ($205 \times 205 \times 6$) combinations.

When we examined the size of combinations, CorCoCCM produced the smallest sizes (28, 32, and 17 for RCO, RPB, and RGR, respectively for both 2-way and 3-way). For CaCoCCM and CoSysCo, they produced the same sizes for both 2-way and 3-way: 70, 70, and 42 for RCO, RPB, and RGR, respectively, which are almost 2.5 times larger than those produced by CorCoCCM. This is because both control techniques consider all possible elements for coordination analysis. Because CoSysCo does not support separate system and coordination level analyses and requires users to go through the same set of states for both levels, it produced the same results as the system level.

To investigate how the number of combinations and size of combinations can affect the time required to analyze ONBs, we measured the ONB analysis time, which includes both system and coordination level analyses. The results of the analysis time for each technique for 2-way and 3-way combinations for each task are shown in Table IV. Overall, we observe that the ONB analysis with CorCoCCM takes the shortest amount of time when compared to the control techniques (CoSysCo and CaCoCCM). Further, across all approaches, the ONB analysis using 2-way combinations took less time when compared to 3-way combinations. The results also show that despite having higher numbers of combinations than CoSysCo, CorCoCCM took less time. This is because the size of combinations is much smaller than that of CoSysCo. Further, the results show that CaCoCCM takes much longer time than CoSysCo and CorCoCCM. For example, in RPB, using 3-way, CaCoCCM requires 2103.98 hours for analysis (estimated from the random sample) whereas CorCoCCM and CoSysCo require 3 and 3.75 hours, respectively.

2) RQ2 analysis: While reducing time and effort is important, if such reductions miss important ONBs, the approach would not be beneficial. To investigate whether the reduction achieved by our approach compromises the ONB detection capability, we collected the number of ONBs found at the system level and coordination level shown in Table V. The results indicate that CoSysCo missed 1 ONB each at the system and coordination levels when compared to CorCoCCM in RCO and RPB. We attribute this to the lengthy combinations in CoSysCo. In the case of CaCoCCM, because the number of combinations the user needs to examine is prohibitively high, we analyzed a random sample of combinations. Moreover, because the combinations of CaCoCCM covers the entire combinations of CorCoCCM, the number of ONBs found using CaCoCCM must be at least as many ONBs as those detected by CorCoCCM. In our study, the random sample we analyzed always found a smaller or equal number of ONBs as CorCoCCM. Hence, we reported the same number of ONBs for CaCoCCM and CorCoCCM. The results also show that in the case of RCO and RPB, 3-way combinations were able to find higher numbers of ONBs at both system and coordination levels for all three techniques when compared to 2-way combinations. In the case of RGR, all three techniques found the same number of ONBs at both system and coordination levels for both 2-way and 3-way combinations. We speculate that the simplicity of the coordination task affected this outcome.

TABLE V
TOTAL NUMBER OF ONBS FOUND AT SYSTEM LEVEL (SYS LVL) AND COORDINATION LEVEL (COORD LVL)

Req.	Level	CoSysCo		CaCoCCM		CorCoCCM	
		2-way	3-way	2-way	3-way	2-way	3-way
RCO	Sys	16	16	**17**	**17**	**17**	**17**
	Coord	7	8	8	**9**	8	**9**
RPB	Sys	16	16	**17**	**17**	**17**	**17**
	Coord	9	10	10	**11**	10	**11**
RGR	Sys	**8**	**8**	**8**	**8**	**8**	**8**
	Coord	**14**	**14**	**14**	**14**	**14**	**14**

V. DISCUSSION

In this section, we discuss our results and some implications of the results including the limitations of our work.

CorCoCCM Results: The results show that our approach, CorCoCCM, reduces the amount of time and effort required to perform ONB analysis for multi-robot coordination tasks, without losing its ability to expose ONBs. Our technique, when compared to CoSysCo, reduced the ONB analysis time by ≈ 1 hour and exposed a higher number of ONBs. When compared to CaCoCCM, our approach exposed the same number of ONBs but reduced the time required for ONB analysis by 90%. When compared to the control techniques, our approach eases the manual analysis with a smaller size of combinations. In particular, the size of combinations of states generated during the coordination level analysis by CorCoCCM is 2.5 times smaller than the size of combinations of control techniques, thereby making ONB analysis easier.

CorCoCCM also provides a means of performing thorough analysis by facilitating separate analyses for system and coordination levels. Hence, when multiple experts and stakeholders perform analysis using CorCoCCM, they can reduce the time and effort further because analysis can be done in parallel.

System Level and Coordination Level Reliability: In our study, we identified ONBs after performing both system and coordination level analyses, but we did not correct the ONBs found from the system level analysis before we perform the coordination level analysis. This means that we do not know whether fixing ONBs found at the system level can possibly prevent from occurring ONBs at the coordination level. Thus, to have a better understanding of the benefits of our approach, we investigated the following question: "If we make the systems reliable by correcting ONBs found at the system level, are coordination tasks performed by them reliable?".

To investigate this question, we analyzed the ONBs found from the system level analysis and corrected the ONB problems. For example, in RCO, we found the following ONB: "Treel system that controls the movement of a robot moving forward when infra red sensors detected an object". We addressed this ONB by preventing its occurrence by adding a new requirement which states "When infra red sensors detect an object, the treel system shall stop". By addressing all other ONBs at the system level, we are able to ensure that the systems participating are always in a reliable state as they can never reach or they can recover from an undesired state that results in ONB.

After ensuring that the system states of all systems are reliable, we analyzed combinations of states of coordination entities for ONBs. Our results show that even after each system is in a reliable and safe state, we found 6 ONBs in RCO, 5 ONBs in RPB and 10 ONBs in RGR. Examples of ONBs we found even when each system is in a safe state are as follows:

1) RCO: Robots moving in opposite directions when held together an object.
2) RPB: Robots placing an object on the block and knocking over the block when changing their direction.
3) RPB: One of the robots blocking the view and IR sensors of the other robot, resulting in no movement or crash.
4) RGR: The message about the destination is lost, and a robot crosses the destination.

Hence, we can conclude that *the reliable systems do not necessarily perform coordination tasks reliably.*

Limitations: Although our approach offers several benefits and the empirical results are promising, our approach has some limitations that need to be addressed. One limitation involves the size of combinations. As the number of components and coordination entities increases, the size of combinations can be very large, which can hinder the manual analysis process and can make it error-prone. We plan to address this limitation using association rule mining to generate sets of dependent components. Another limitation is that our approach is not completely automatic and requires manual analysis. We have tried to automate the model generation and model entity identification process using NLP [16], but the results are not very promising. We plan to address this limitation in part in the

future by collecting more requirements documents and creating more training data.

VI. Threats to Validity

One threat to validity in our study is the process of identifying ONBs and measuring time. This process has been performed by a single expert, which can be subjective and the results can be biased by personal knowledge and experience. This threat can be mitigated as multiple experts and stakeholders are involved during the analysis. Further, the requirements we used in this study came from research projects, which are less complex compared to industrial projects. Thus, the results we obtained might not be applicable to the industrial applications. We plan to address this threat by utilizing industrial projects in our experiment.

VII. Related Work

To date, many researchers have worked on analyzing multi-robot coordination [6], [7], [17], [18]. Some researchers have concentrated on the communication aspect of the coordination [6], [18]. For example, Sheng and Yang [18] proposed a stochastic petri-net based method for peer to peer communication among robots to improve the coordination of the robots. In this approach, the authors consider potential issues caused by communication and address them via petri-nets. Other researchers have proposed fault tolerant architectures for multi-robot coordination [6], [17]. For example, Parker [17] proposed a software architecture called Alliance, which is a behavior based architecture for multiple heterogeneous robot coordination. The architecture has a fault tolerant mechanism, which aids the robots to adapt themselves by detecting the effect of their own actions as well as other robots in the team. Further, some researchers have proposed a formal analysis for multi-robot coordination [7], [19], [20]. For example, Alur et al. [19] proposed a formal modeling approach for analyzing and designing a group of robots that need to coordinate to achieve the specified goals. The approach analyzes stategies for communication and control among robots. Jeyaraman et al. [20] proposed a formal modeling and validation technique using Kripke models. The approach generates temporal logic from Kripke models and uses the temporal logic for model checking on the predicted behavior of the system.

Despite the advantages offered by these approaches, none of them considered ONB analysis or provided ways to support ONB knowledge acquisition. In our proposed approach, we address this limitation and provide a way to acquire knowledge of ONBs at the early stage of software development (e.g., requirements engineering or design phase).
to perform ONB analysis and exposes hidden ONBs that can propagate defects to the later phases of software development.

VIII. Conclusion and Future Work

In this paper, we proposed CorCoCCM to expose ONBs in tasks that require multiple system coordination. The proposed approach facilitates the separate analyses of coordination tasks at two different levels (system and coordination) to expose ONBs. To evaluate our approach, we performed an empirical study using three coordination task requirements. Our results show that CorCoCCM reduces human effort and time required

Although the empirical results with CorCoCCM are promising, there are some limitations discussed in Section V. In our future work, we plan to address these limitations. Further, when we analyze ONBs, we would like to consider different types of communications (e.g., inductive telemetry, and radio communication) and environmental contexts to make the system robust.

Acknowledgment

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

References

[1] N. G. Leveson, "System safety in computer-controlled automotive systems," *SAE transactions*, vol. 109, no. 7, pp. 287–294, 2000.

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

[3] A. T. Vemuri, M. M. Polycarpou, and S. A. Diakourtis, "Neural network based fault detection in robotic manipulators," *IEEE Trans. Robot. Autom.*, vol. 14, no. 2, pp. 342–348, Apr 1998.

[4] S. Verma, S. Lozito, K. Thomas, and D. Ballinger, "Procedures for off-nominal cases: Very closely spaced parallel runway operations," in *DASC'08*, 2008, pp. 2.C.4–1–2.C.4–11.

[5] K. Madala, H. Do, and D. Aceituna, "A combinatorial approach for exposing off-nominal behaviors," in *ICSE'18*, 2018.

[6] Z. Yan, N. Jouandeau, and A. A. Cherif, "A survey and analysis of multi-robot coordination," *IJARS*, vol. 10, no. 12, p. 399, 2013.

[7] E. Matson and S. Deloach, "Integrating robotic sensor and effector capabilities with multiagent organizations," in *IC-AI'04*. Citeseer, 2004.

[8] D. Bozhinoski, D. D. Ruscio, I. Malavolta, P. Pelliccione, and I. Crnkovic, "Safety for mobile robotic systems: A systematic mapping study from a software engineering perspective," *Journal of Systems and Software*, vol. 151, pp. 150 – 179, 2019.

[9] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence, "IPOG/IPOG-D: efficient test generation for multi-way combinatorial testing," *JSTVR*, vol. 18, no. 3, pp. 125–148, 2008.

[10] M. Dorigo, E. Tuci, R. Groß, V. Trianni, T. H. Labella, S. Nouyan, C. Ampatzis, J.-L. Deneubourg, G. Baldassarre, S. Nolfi, F. Mondada, D. Floreano, and L. M. Gambardella, "The swarm-bots project," in *Swarm Robotics*. Springer Berlin Heidelberg, 2005, pp. 31–44.

[11] F. Ducatelle, G. A. Di Caro, and L. M. Gambardella, "Cooperative self-organization in a heterogeneous swarm robotic system," in *GECCO '10*. New York, NY, USA: ACM, 2010, pp. 87–94.

[12] T. Stirling, J. Roberts, J. C. Zufferey, and D. Floreano, "Indoor navigation with a swarm of flying robots," in *ICRA'12*, May 2012, pp. 4641–4647.

[13] A. Saxena, C. S. Satsangi, and A. Saxena, "Collective collaboration for optimal path formation and goal hunting through swarm robot," in *Confluence'14*, Sept 2014, pp. 309–312.

[14] G. Baldassarre, D. Parisi, and S. Nolfi, "Coordination and behaviour integration in cooperating simulated robots," in *SAB'04*, 2004.

[15] L. Yu, Y. Lei, R. N. Kacker, and D. R. Kuhn, "Acts: A combinatorial test generation tool," in *ICST'13*, 2013, pp. 370–375.

[16] K. Madala, D. Gaither, R. Nielsen, and H. Do, "Automated identification of component state transition model elements from requirements," in *RE'17 Workshops, AIRE'17*, Sept 2017, pp. 386–392.

[17] L. E. Parker, "Alliance: An architecture for fault tolerant multirobot cooperation," *IEEE Trans. Robot. Autom.*, vol. 14, no. 2, pp. 220–240, 1998.

[18] W. Sheng and Q. Yang, "Peer-to-peer multi-robot coordination algorithms: petri net based analysis and design," in *AIM'05*, July 2005, pp. 1407–1412.

[19] R. Alur, J. Esposito, M. Kim, V. Kumar, and I. Lee, "Formal modeling and analysis of hybrid systems: A case study in multi-robot coordination," in *FM'99*, J. M. Wing, J. Woodcock, and J. Davies, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 212–232.

[20] S. Jeyaraman, A. Tsourdos, R. Zbikowski, and B. White, "Formal techniques for the modelling and validation of a co-operating uav team that uses dubins set for path planning," in *Proceedings of the 2005, American Control Conference, 2005.*, June 2005, pp. 4690–4695 vol. 7.