Detection and Verification of a New Type of Emergent Behavior in Multiagent Systems

Fatemeh Hendijani Fard*, Behrouz H. Far*
* Department of Electrical and Computer Engineering, University of Calgary, Calgary, Canada
{fhendija, far}@ucalgary.ca

Abstract—The verification of Distributed Software Systems (DSS) and Multi agent systems (MAS) has taken a special attention due to the growing demand of having DSS in this decade. MAS and DSS are a class of software in which functionality or control is distributed. This may cause components (agents) to emerge an unexpected behavior in their runtime, which was not seen in the requirement and design. This is known as emergent behavior of components. The cost of detecting and fixing of such problem is much more valuable compared to fix them after deployment. Therefore, in this paper a new type of emergent behavior that can happen in MAS is investigated. A method for verification of this type of emergent behavior is proposed followed by an algorithm. This type of emergent behavior can not be detected with the existing methods of emergent behavior detection. This type of emergent behavior focuses on one component when it misses the information about the senders of the same message from different components. The contribution of this work rather than investigating this type of emergent behaviors is on its verification method and also proposing a solution to fix it. The details are shown through a case study on MaSE artifacts which is an Agent Oriented Software Engineering methodology.

I. INTRODUCTION

The verification of Multi Agent Systems (MAS) which are a sub-class of Distributed Software Systems (DSS) has taken a special attention due to the growing demand of having MAS in this decade.

One of the issues in MAS is the lack of central control that can lead to unexpected behaviors in run time which was not seen in the requirements of the system [1]. These unexpected behaviors known as emergent behaviors may cause critical and irreparable damages to the systems [2]. Studies show that detecting and fixing these behaviors in the analysis and design phase has a great advantage in terms of saving costs, time and effort [1]. It is stated that “Detecting the causes of faults early may reduce their resulting costs by a factor of 100 or more” [3].

Moreover, scenario based approaches like UML Sequence Diagrams (SD) and Message Sequence Charts (MSC) are widely used in the requirement and design phases of DSS. Scenarios are used in DSS due to some characteristics like easiness, having semantics and the ability to construct verification models from them. Therefore, many researches are dedicated to detect the emergent behaviors in the requirement and design phases of scenario based software systems. Most of the researches use algorithms based on behavioral modeling for the detection of emergent behaviors. The synthesized behavioral models illustrate the components’ behaviors and are later used for verification of system behavior against the models from the requirements and design. Although the model checking has advantages, there are lots of problems with the behavioral modeling and system verification with model checking. State space explosion especially in DSS [4, 5], the expertise required for modeling or specifying requirements using the existing languages [6], and mostly working with synchronous communication style are among problems found for these approaches.

Therefore, in this paper we focus on a new technique for the detection of emergent behavior in MAS and DSS scenario based software systems. Furthermore, a new type of emergent behavior is introduced here. The classification of emergent behavior types is important to use them as a marker for suggesting solutions to fix them. This is one issue which is mostly ignored in the existing approaches. The existing approaches do not show the exact cause or point of happening the emergent behavior [7], and therefore, they suggest no clue to the designer for fixing the detected emergent behavior. In this paper, the type of emergent behavior introduced is in agent based systems. The problem arises when one agent is missing the information about the senders of same messages. This type of emergent behavior cannot be detected with the existing methods. The details and examples are shown in next sections. The contributions of this work are:

- Introducing a new type of emergent behavior type.
- Proposing a new technique and algorithm for detection of the explained emergent behavior type.
- Suggesting solutions to fix the problem.

This paper is organized as follows: related works are reviewed in section II. In section III the new type of emergent behavior is introduced. The proposed technique and the devised algorithm are discussed in section IV explained in detail through a case study. Section V is dedicated to the discussion followed by conclusion and future works in section VI.

II. RELATED WORKS

A. Emergent Behaviors

Emergent behavior is defined as the unexpected behaviors of software components in the execution time, while it was not seen in their requirement specification [1]. Emergent behaviors in some works are referred to as implied scenarios i.e. when integrating all of the scenarios of the system (e.g. in the form of state machine), they may imply a new scenario. Emerging new behaviors in DSS is more probable because there is lack of central control in these systems. This reason causes the components to have
a local view of the system. Consequently, they may start with one scenario of the system and continue in another scenario in a shared state [8]. This state in some researches is referred to as identical states [9].

One approach for model checking in the requirement is Alur et al. methodology [10]. His works define a detailed explanation of the model checking of MSC and high-level Message Sequence Chart (hMSC). MSCs can be considered as an early formal model for the system that specifies system’s behavior. All of the linearization of this model is then checked against an automaton which is defined by the message alphabet \( \Sigma \), words, languages, states, and transitions which show the behavior of each process or component.

Whittle and Schumann et al. propose a methodology which uses Unified Modeling Language (UML) notations and investigates the conflicts in translation between UML notations. They define a methodology supported by a tool to synthesize statecharts from Sequence Diagrams (SD), class diagrams, and Object Constraint Language (OCL) specifications. Their main focus is on using their methodology for Agent Based Systems (ABS) [11].

Ben-Abdallah explains the non-local branching choice which causes the emergent behavior in system. This problem may arises when there is a branch in the MSC specification, and the first events, in the different branches, are sent by different components [12].

Considering the non-local branching choice, Muccini inspects its effect on emerging implied scenarios and explains the reason of an implied scenario generation. When the state machines are synthesized from scenarios, a set of behaviors are presented by state machines which were not in the scenarios [13]. In the non-local choices, different processes send first events in different branches. This is detected by “augmented behavior” of processes in the non-local choice in their algorithm. When processes share the same augmented behavior in the non-local choice, their interaction generates an implied scenario.

Song et al. methodology is quite different with the other approaches. They explain the detection of implied scenarios when using UML. They generate two graphs named specification and implementation graphs. By matching these two graphs, the implied scenarios and the exact cause of its emergence in the specification are identified [7].

Uchitel et al. provide a method and tool for detection of implied scenarios. Their algorithm builds Labeled Transition Systems (LTS) for behavioral modeling of MSC and hMSC [14]. The work is also extended by Letier et al. to detect input-output implied scenarios. These scenarios can’t be detected by other approaches developed for implied scenarios [15].

Kumar et al. discuss the problems with main researches in this field. He discusses that many researches are not implementable and amendable, or don’t show correctness. They use Message Sequence Graphs (MSG) and FSM and define a reduced transition system to detect implied scenarios and model checking. They develop a complete method for this problem claiming the correctness and ability of implementation of their work [16].

B. Message Sequence Charts

Message sequence charts are commonly used in recent years to specify system requirements and is used to capture and validate early requirements and design specially for scenario specification of DSS [17]. MSCs are a visual description of requirements of a software system [18]. Like a SD, a MSC shows the interactions of components of a system. Each vertical line shows a component and the interacted messages between them are shown with arrows. An MSC formally is defined as [14] a set \((E, L, C, \mu, <, \text{Map})\) where:

- \(E\) is the set of send and receive events.
- \(L\) is a finite set of labels.
- \(C\) is the set of components (agents).
- \(\mu\) is the mapping of events to labels and components.
- \(<\) is the set of total orders on the events and \(\mu\).
- \(\text{Map}\) maps the send events to receive ones.

Figure 1 shows two examples of MSCs.

III. NEW TYPE OF EMERGENT BEHAVIOR

There are some reasons named in the literature as the causes of emergent behaviors. Some of the causes are local view of the components from the whole system [19], having identical states [9, 19], not knowing the conditions of each event to satisfy the conditions [14], non-local branching choices [12], and unenforceable orders [7]. Most of these works use the behavioral modeling as the verification method. However, this model does not consider the interaction between agents individually, while this is an important issue in MAS. By “the interaction between agents individually” we mean that in behavioral modeling the behavior of each agent is investigated for the detection of emergent behavior. Also in system level, the behavior models of all agents when they are executed parallel are investigated. However, whether who is the sender or receiver of a message is not considered. Therefore, it is not taken into account that which agent causes a state transition for another agent (Each event for an agent in an MSC is considered as a state.) But this should be considered when investigating the emergent behaviors in multi agent systems. Also, considering this point can make the detection of emergent behaviors easier and with less complexity. Moreover, this point can help suggesting solutions for the detected emergent behavior. Even considering the anonymity of agents, in FIPA agent architecture the sender/receiver are mandatory elements of message envelope [20].

One of the problems that can be caused by the local view of agents in the agents’ interactions is missing the information about senders of same messages that we refer to it as issue I1. This causes the agent to go in the same state in two different scenarios. The same state in different scenarios is also known as identical states [1]. In behavioral modeling, the identical states are integrated into one state machine and the next states are branches of them which are later converted to deterministic finite state machines. The problem that may arise in this situation is when issue I1 happens. Here, the agent who is the receiver of same messages has the local view and shared states in two scenarios. But the agent has two different choices because of missing some information about the senders, not because of indeterministic behavior as mentioned in [1]. Therefore, the shared states are not real identical states for that agent and should be treated differently.
One such example is shown in Figure 1 that illustrates the interactions between three agents in two scenarios MSC 1 and MSC2. For agent Aw, there are three messages sent/received in each scenario. The first two messages for Aw are the same messages (M1 and M2), however, the third message are different. The behavioral modeling approaches consider the first two states of Aw the same states. What issue 11 mentions in this situation is that these two states should not be considered as identical states; because the senders of same messages, namely M1, in the two MSCs are different.

![Figure 1. Two MSCs with shared states for agent Aw causing emergent behavior](image)

The proposed method for detecting the type of emergent behavior that we introduced in this section is explained in detail in next section.

IV. PROPOSED METHOD

In all parts of this paper the words component and agent may be used interchangeably, while by both of them we mean the word “agent”.

A. Definitions

**Matrix:** We build one \( n \times n \) matrix for each scenario (MSC) where \( n \) is the number of components in the system. Matrix related definitions are given below.

**Definition 1 (Send matrix):** Matrix \( S \) is defined as the send matrix. MSC\(_k\) is the \( k^{th} \) MSC of the system. For each MSC\(_k\), if there is a message sent from component \( C_j \) to component \( C_i \) then we put the order of message as appeared in the scenario as the entry in the \( S \) matrix (\( S_{ij} \)), otherwise \( S_{ij} = 0 \). If there is more than one message sent from one component to another, the numbers are separated with a comma in that entry. The \( S \) is the send matrix related to MSC\(_k\).

Likewise, Definition 2 shows the receive matrices:

**Definition 2 (Receive matrix):** Matrix \( R \) is defined as the receive matrix. MSC\(_k\) is the \( k^{th} \) MSC of the system. For each MSC\(_k\), if there is a message received by component \( C_i \) from component \( C_j \) then we put the order of message as appeared in the scenario as the entry in the \( R \) matrix (\( R_{ji} \)), otherwise \( R_{ji} = 0 \). If there is more than one message received for one component, the numbers are separated with a comma in that entry. The \( R \) is the receive matrix related to MSC\(_k\).

We put the order of message as appeared in the scenario to keep track of the order of the messages sent or received in a scenario. For example, if component \( C_2 \) sends a message to component \( C_i \) and this is the third message send in the related scenario, we insert 3 for it or we will have \( S_{32} = 3 \) in the related send matrix.

In this context each matrix is informative, since it shows the components, the relationship between them, and the order of messages. The only information missing is the message labels (content).

Since the synchronous communication style is focused here, the send and receive matrices are equivalent. Considering the synchronous communication and send matrices, other definitions based on these basic definitions (Definition 1 and Definition 2) can be outlined. Other definitions help investigate the emergent behavior detection process and show the specific information about components, e.g. state transition vectors, state transition sender vectors, and component communication vector. Each definition will be used in further techniques and algorithms for the detection of emergent behaviors.

In the finite automata each message can transmit the component from one state into another. This transition can be extracted from the matrices defined here. The advantage of using these matrices is that we can define which component of the system has caused the state transition for each individual component. The state and state transition formal definitions are explained by Alur [2].

For having the state transition sender vectors for an individual component \( C_i \), the send matrix \( S \) should be considered. The entities of \( S \) matrix in the row and column \( i \) are the related message numbers related to \( C_i \). The order of the numbers in the row and column \( i \) shows the order of states for that component. In the case that the component had sent a message to itself, two states are considered. The row number other than \( i \) show the senders of the messages for component \( C_i \).

For each MSC\(_k\) which shows the \( k^{th} \) MSC of the system, the related \( n \times n \) send matrix is shown with \( S \).

For each component \( C_i \) in matrix \( S \) related to MSC\(_k\) the entities \( s_{ij} \) show the entities in row \( i \) and column \( j \) where \( i \leq s \leq n \). This means the message numbers that component \( C_i \) has sent to other components and to itself. Also the entities \( s_{ji} \) show the entities in column \( i \) and rows \( r \) where \( i \leq r \leq n \). This means the message numbers that component \( C_i \) has received in the related MSC\(_k\).

**Definition 3 (Component send communication vector):** Vector \( v_{sik} = (e_{i1}, e_{i2}, ..., e_{in}) \) shows the vector of row \( i \) in matrix \( S_k \) and \( n \) is the total number of components. When there is more than one number in one entry of the \( S \) matrix, the element is considered as a compound element containing more than one number in the vector \( v_{sik} \). It means for example if there is entry \( s_{i1} = 2, 3 \) in matrix \( S_k \) then in the vector \( v_{sik} \) element \( e_{i1} \) will be \( e_{i1} = (2, 3) \). This vector shows all of the communications of component \( C_i \) to other components in the \( k^{th} \) scenario (MSC) where the component \( C_i \) is the sender of the messages.

**Definition 4 (Component receive communication vector):** Vector \( v_{rik} = (r_{i1}, r_{i2}, ..., r_{in}) \) shows the vector of column \( i \) in matrix \( S_k \) and \( n \) is the total number of components. Similar to Definition 4, when there is more than one number in one entry of the \( S \) matrix, the element is considered as a compound element containing more than one number in the related vector \( v_{rik} \). This vector shows the relations and communications of component \( C_i \) to other components in the \( k^{th} \) MSC where the component \( C_i \) is the receiver of the messages.

The state vector of each component \( C_i \) in MSC\(_k\) is defined as:
Definition 5 (Component state vector): The states of each component $C_i$ in $k^{th}$ MSC is shown by a vector $states_{ik}$ which is the ordered vector of the entities of $v_{ik}$ and $v_{ak}$ where the zero entities are omitted and the elements are in ascending order. In the case that a component had sent a message to itself, the same numbers in the $v_{ik}$ and $v_{ak}$ are added distinctly to the states vector. This shows in this case two states are considered for this component by sending a message to itself: one for sending and one for receiving the message. The number of the entities in vector $states_{ik}$ shows the total number of events on component $C_i$ in MSC $k$ or the projection of events of $k^{th}$ MSC on $C_i$ (MSC$_k|C_i$). In vector $states_{ik} = (s_{t1}, s_{t2}, ..., s_{t|k|}, s_f)$ each element shows the states of component $C_i$. The transitions between the states are by messages sent or received to by component $C_i$. The last element $s_f$ shows the accepted state of component $C_i$ in MSC $k$.

Note that since each element of the state vector $states_{ik}$ matches an element in $v_{ik}$ or $v_{ak}$, it is obvious which of the entities $s_{tp}$ in the related $S$ matrix it matches. Therefore, we know that each state matches either a send event or a receive one. Also we know which component has sent the message related to that receive event of component $C_i$, and vice versa, by tracking the number of that element in each of the $v_{ik}$ or $v_{ak}$ vectors.

To clarify these definitions, consider the message sequence chart in Figure 2 with three components.

Definition 6 (State transition sender vector): The state transition sender vector for component $C_i$ in MSC $k$ is shown by $Sender_{ik} = (Snd_{t1}, Snd_{t2}, ..., Snd_{t|k|})$ where each element shows the sender for a message that changes the state for component $C_i$. Each element of $states_{ik}$ has a matching either in $v_{ik}$ = ($e_{i1}$, ..., $e_{ik}$) or in $v_{ak}$ = ($f_{i1}$, ..., $f_{ik}$). Therefore, the senders of the messages for each state are either component $C_i$ (if the element in $states_{ik}$ has a matching in $v_{ik}$) or are the component $C_n$ (if the element in $states_{ik}$ has a matching with the $w^{th}$ element in $v_{ak}$).

Simply explaining, this vector shows which component is the sender of each event for component $C_i$ in MSC $k$.

Continuing our example in Figure 2, the state transition sender vector for component $C_2$ in Example-MSC 1 is $Sender_{21} = (C_1, C_2, C_3, C_3)$.

B. Proposed Method

The proposed method for the detection of this type of emergent behavior includes following steps:

1. Transferring the MSCs to send matrices.
2. Extracting the state and state transition sender vectors for each agent from all MSCs.
3. Detecting the agents that have no emergent behavior and omit them from processes in next steps.
4. Find the shared states for each agent in different MSCs and compare the state transition sender vectors for each agent. (Apply Algorithm 1, see section IV part C).
5. Report the detected agents and their states potential for emerging a new behavior to the system designer.

First the send/receive matrices and related vectors using the definitions are extracted. The detection of agents with no emergent behavior (step 3) comes next to reduce the complexity of later steps and make the emergent behavior detection more efficient. The algorithm used in step 3 is previously devised in [21]. The detection of emergent behavior is done with Algorithm 1. A report is generated indicating suggested solutions and the states that each agent may show an emergent behavior. The report can help the system designer fix the problem with exactly referring to the design flaws. All the steps and the process are explained through a case study in next section.

C. Algorithm

The algorithm for the detection of the type of emergent behavior discussed in section III (Algorithm 1 of the fourth step of the proposed method) is given below:

Algorithm 1. Detecting emergent behaviors

Input: states$ = (st_1, ..., st_f)$; Sender$ = (Snd_1, ..., Snd_p)$

Output: The information of $A_i$ state that causes EB

DetectState0EB(data.ReceiveMessage,states$\_s$,Sender$\_s$)

for all agents $A_i$ do

sharedStates$\_s$←FindSharedParts (states$\_s$)

if Successor(st$[f][k]$) ReceiveMessage then

sharedStates$\_s$←Remove (states$[i][k]$)

end if

for states$ \_s$, sharedStates$ \_s$ do

DiffStateMSC$ \_s$ ←FindDifferentSender (Sender$ \_s$, sharedStates$ \_s$)

Print (DiffStateMSC)

end for

end for

In the first step of the algorithm, all the state vectors for each agent are compared to each other to find their shared parts. This shows the shared states of one agent in different MSCs. Then the related state vectors must be checked for their successor of their last shared state. If the successor of the last state is a send message, the state vector will be checked for emergent behavior; otherwise, it will be omitted. This is a critical point in the detection of this type of emergent behavior. An agent can show an
unexpected behavior by sending a message to other agents. In the situations that the agent is a receiver it can’t cause an emergent behavior. This is proven when using formal automata and applied to our method, too.

The next step compares the remaining state vectors and their shared parts. The states are compared for their sender vectors. If the shared states have the same senders there will be no problem. But if shared states have different senders have the potential to emerge a new behavior in that states. In other words, these emergent behaviors are those behaviors caused when for one agent, the sender of same messages are more than one type of agent. The detected states and their related MSCs are printed in the report. The report also contains the sender of messages for each state. Therefore, it suggest a solution for the system designer, either to add a variable showing the sender of messages for the detected agent or redesign reported MSCs to fix the problem. The revised designed can be checked again for verification.

D. Case Study

As a case study a greenhouse multiagent system is considered to explain the proposed method. This system consists of three different types of agents: Temperature balancing agents (At), Water control Agents (Aw), and Mineral control Agents (Am). The agents receive environmental information from sensors, connect to data and knowledge bases and analyze the information to perform the best task. The agents are supposed to perform autonomously and interact with each other in order to keep the plants in the best situation. Some of the functional requirements of this system are: Managing water resources, minerals, and the normal temperature for the plants, Interacting with other agents to save energy, Monitoring the changes in the environment, and Saving changes and decisions in a Database. The system is designed using the MaSE methodology. The detailed MaSE artifacts are not shown because of page limitation.

MaSE has sequence diagrams instead of MSCs. The SDs are transformed to MSCs using the technique explained in [22]. The following shows two MSCs of this system in Figure 3 and Figure 4.

![Figure 3. MSC1-Mineral balancing with mist](image)

Figure 3. MSC1-Mineral balancing with mist

After extracting matrices from the MSCs, the rows and columns related to agent $A_*$ are considered. In the MSCs, we will consider the rows and columns related number to this agent as $i=5$. The next step is to extract the related vectors.

![Figure 4. MSC2-Temperature balancing](image)

Figure 4. MSC2-Temperature balancing

For this agent, the vector $states_i$ shows the associated states vector of $A_i$ in the related MSCs and $k$ shows the $k$-th MSC. The states vectors of $A_i$ are:

For this agent, the state transition sender vectors are shown with $Sender_i$ in the $k$-th MSC:

By applying Algorithm 1, it is found that $A_i$ has shared states in these two MSCs for its first three states. For these states the sender vector related to shared states for MSC1 is $\{A_{m}, A_{w}, A_{w}\}$ while the sender vector for shared states in MSC2 is $\{A_{w}, A_{w}, A_{w}\}$. By comparing the sender vectors the results show a difference in the senders. The difference of the senders between these vectors is the sender of first message. It means in MSC1, the agent $A_i$ is the sender of the message that brings $A_i$ into state $s_{i1}$ while, for this state in MSC2, the agent $A_i$ is the sender of the message that brings $A_i$ into state $s_{i2}$. Since senders are different and the next event is a sending a message by $A_i$, this agent may emerge a new behavior in this state. This is caused by missing which agent is the sender of the same messages in the two MSCs.

Fixing the detected emergent behaviors can be done by adding arguments to the concurrent task and conversation steps. Therefore, the agent knows who sends the message and how it should continue the other tasks it is involved in. By adding an argument, the states will not be the same states for the agents in the execution time and therefore no unexpected behavior could happen because of missing information about the sender of a message. The other solution is revising the designs by the system designer and checking them again for this type of emergent behavior.

Fixing the detected emergent behavior in our case study is shown in Figure 5 for the concurrent tasks of “Receive request” of the water control agent. $Aw$. It keeps an argument $S$ associated to Sender of a request in its states.

![Figure 5. Concurrent Task Diagram for Aw](image)

Figure 5. Concurrent Task Diagram for Aw
V. DISCUSSION

In this article a new type of emergent behavior is introduced which cannot be detected with the existing methods. The reason is that the existing approaches consider the parallel behavioral modeling and ignore the individual interaction between agents as discussed in section III. This type of emergent behavior arises because one agent misses the information about the senders of same messages in its identical states in two different scenarios, when senders are of different types and also it is the sender agent after these states.

The technique proposed here uses relationship matrices. While the matrices may contain many zero numbers, our technique does not consider the zero values and just the non-zero values are added to different vectors. Therefore, saving time and memory for the method is not a big challenge because just the non-zero numbers are taken into account. Also the algorithm we devised in [21] omits the agents that will not have emergent behavior. This approach also will help to reduce the complexity.

VI. CONCLUSION AND FUTURE WORK

The introduced type of emergent behavior and the proposed method in this paper can help cover the scalability problem in multi agent systems. Also the classification of emergent behavior types and focusing on them will help suggesting solutions to fix them rather than just detecting the emergent behavior. Based on our knowledge, this approach is ignored in the existing works.

For the future work, we are planning to expand the algorithm that can work with different send and receive matrices. Therefore, it also can investigate the emergent behaviors in the effect of service degradation in the case of lost or corrupted messages.

ACKNOWLEDGMENT

This research was partially supported by a grant from Natural Sciences and Engineering Research Council of Canada.

REFERENCES