A New Approach for the Detection of Emergent Behaviors and Implied Scenarios in Distributed Software Systems

Extracting Communications from Scenarios

Fatemeh Hendijani Fard and Behrouz H. Far

Department of Electrical and Computer Engineering, University of Calgary, Calgary, Canada

1 RESEARCH PROBLEM

An approach to specify the requirements and design of a Distributed Software System (DSS), which is mostly used in recent years, is describing scenarios with visual artefacts, such as, UML Sequence Diagrams and ITU-T (ITU-T UNION, 2004) Message Sequence Charts (MSC) and High level Message Sequence Charts (hMSC) (Krüger, 2000). Scenarios describe system's behavior and define the components and their interactions. Each scenario determines a partial behavior of the system. Hence, the restricted view of the components in each scenario and distributed functionality and/or control in DSS, may result in inconsistency in the system behavior.

One problem that arise in scenario based Distributed Software Systems is emergent behaviors or implied scenarios that occur because of restricted view of one or more components. Emergent behaviors are known as unexpected behaviors that components show in their execution time (Uchitel, 2003; Bhateja et al., 2007). However, this behavior was not defined in their designs. This unexpected behavior may imply a new scenario to the system, and can result in considerable cost and damage (Alur et al., 2005). Therefore, emergent behaviors should be detected in the early phases of software development to prevent damage or cost after deployment. The detected emergent behaviors can be either accepted or denied by the stakeholders. However, they should be detected and discussed, to be added as new designs, or to be specified as negative scenarios that should be avoided (Uchitel et al., 2002).

In our research, we try to devise an automatic methodology to detect the emergent behaviors (implied scenarios) from the designs of the system. We also mean to help the designers for the exact point of the problem in the system and the possible solutions to remove the detected emergent behaviors.

2 OUTLINE OF OBJECTIVES

Many approaches in the literature are defined, for the detection of emergent behaviors in early phases, to save cost of fixing them after deployment. Some of the approaches use formal methods and Finite State Machine methodologies to construct the behavior modeling of the components and verify them against some properties.

Some of the issues with many methodologies under this category are performance, their cost, the amount of time for defining the constraints and rules, besides expertise and knowledge required for the applications and language notations or techniques (Holzmann and Smith, 2002; Iglesias, 2009; Uchitel, 2009; Briand, 2010). Furthermore, in the requirement model checking of scenarios, these approaches cannot identify the exact location of the scenario specification causing errors (Song et al., 2011).

Some of the problems we have found during our study are:

P1: The process of constructing behavioral models is complex and hardly scalable. Therefore, it needs special algorithms and tools to model components' behavior besides the emergent behavior detection. The process of behavioral modeling is time consuming and complex (Mousavi, 2009).

P2: The existing methods using behavior modeling are message dependent. This requires a great time and effort to verify the specifications if system requirements change, e.g. adding a new component or modifying interactions between the existing components. In this case, the whole process should be done from the scratch. Besides, the message dependency in this level requires domain expertise to annotate the model or specify proper specifications (Chaki et al., 2005)

P3: While system requirements especially in scenario based systems show the interaction of all types of components of the system, they can't show

Hendijani Fard F. and H. Far B..
A New Approach for the Detection of Emergent Behaviors and Implied Scenarios in Distributed Software Systems - Extracting Communications from Scenarios.

this interaction among all instances for each type. Therefore, emergent behavior can still exist between some components of the same type (e.g. all sellers in an online auction system are of the Seller type); but existing research cannot handle this.

P4: Differentiating between send and receive messages is not considered in many researches (Song et al., 2011), or needs identifying specific definitions to recognize between send and receive messages between the components. While in real world, send messages are not received at the moment they are sent. This makes a flaw between detecting emergent behaviors in requirement phase and what really happens in system execution.

Specifically, we have the following research objectives in our research:

O1: Classifying emergent behavior types in DSS.

O2: Devising a message content independent technique for detection of a subset of classified emergent behaviors of G1 in DSS addressing the P1-P4 problems.

O3: Implementing a tool that can find emergent behaviors in DSS and providing solution to the research questions.

3 STATE OF THE ART

There is a lot of research that transform the MSCs or SDs to various versions of state machines for the detection of implied scenarios. Some research present theorems on the complexity and decidability of implied scenario detection in different specifications, and in various communication channels and styles (Alur et al., 2005; Bhateja et al., 2007). Some other, provide tools to detect and animate the labelled transition systems for implied scenarios (Letier et al., 2005). (Chakraborty, et al., 2010) mentions that some of these works are not amendable or do not show correctness. A common issue that exists in some methodologies is requiring human input. For example in (Mousavi, 2009) the domain expert should fill out some tables to define the semantic causalities among various messages. We try to solve this issue by using message labels instead of message contents, and not using semantic causality. Based on our knowledge, the only different method is generating and comparing two graphs for specification and implementation, which shows the points that an implied scenario can occur, without specifying clues for the designers (Song et al., 2011).

4 METHODOLOGY

In our approach, the components' communications are modeled into interaction matrices, which is inspired by social network analysis (SNA) (Aggarwal, 2011), in which, the communications among individuals are modeled and mined. In SNA, the identities of the communicating individuals are important (may be confidential and hidden in some cases). Likewise, in our model, we try to keep the processes' labels (identities) in the interactions, rather than just keeping the processes' states. A direct advantage of this modeling is saving all the information about the communication of components. This information, which is preserved throughout the extracted vectors, is used as a clue for the designer to examine the consequences of a design decision and fix the detected emergent behaviors. Another advantage of this modeling is detecting warning points in processes' interactions, in terms of, giving the information about the sender or receiver of a message, or hiding this type of information. The warning points can help the designers figure out the possible problems that are effects of a design decision, and provide clues to the designers on how to fix various issues.

This modeling is one step toward the implementation of design decisions that guides the designers to include necessary information in the system designs. The other benefit of our modeling is visualization of components' interactions and their state diagrams – by preserving information about their interactions – that provides visual support for the designers. The warning points and possible problems can be illustrated in various diagrams, which supply the information about the exact state, MSC, and cause of the emergent behavior in the system.

To overcome the problems mentioned in previous sections, we have two main techniques in our proposed methodology. First, we approach the problem by identifying the components that will not show an implied scenario. These components can be omitted from further component level analyses. This will help scalability of the component level implied scenario detection (i.e. analyzing the behavior of each component, without considering the other components' behaviors in the system) (Fard and Far, 2013). Second, we have classified the implied scenarios in various types. Until now, six main categories are specified, based on the literature, and our studies. To the best of our knowledge, this classification is a contribution of our work. The classification of common types of emergent behaviors can lead to better study the detection, reasons behind each problem, and developing solutions for each type of emergent behavior. The six types that we have classified are:

TYPE TP1: Process p may have an implied scenario when it has some shared states in two or more MSCs and it sends one or more messages to different processes in these MSCs. The send action of process p can be in or after its shared states.

TYPE TP2: This type is similar to TP1. In this type, the send actions of process p are a response to its previous interactions with various processes, where these interactions are exactly before or in shared states of process p. This type is important in cases that security or privacy issues should be implemented. The interaction details of process p may be shown in the designs, but should be hidden in the implementations. Therefore, the design and implementation are not equal. This difference, can lead to an implied scenario.

TYPE TP3: Consider that process p is sending messages in MSC M1 and process q is sending messages in MSC M2. The combination of projections of these MSCs is a new scenario M3 that is implied to the system. In M3, which is not in the designs, both processes p and q are sending messages.

TYPE TP4: This class of implied scenario originates from the asynchronous concatenation of MSCs; a case that the processes perform their tasks independently. In other words, a process may proceed to the next MSC, while other processes are still involved in the previous MSCs. Therefore, there is no guarantee that all events in MSC M2 are performed after all events in MSC M1, where M2 is designed to execute after M1 in hMSC. We have specified various subcategories of this type of implied scenario and the situations that may result in having TP4.

TYPE TP5: A sub category of TP4 is known as non-local or branching choice. In this case, different processes can follow different choices according to the hMSC. However, the result is that some processes follow a branch and the rest follow another MSC. Consequently, the result is not in accordance with any of the branching choices in the hMSC. We have devised this type as a separate class of implied scenarios because of the importance of investigating the interactions of processes in branching choices. We have found that, not all of the processes can follow various branches in a branching choice in hMSC. The processes that may behave differently are the ones that start an interaction in those MSCs and do not depend on receiving some messages from other processes to continue their actions. We call these *active processes*.

TYPE TP6: The local visual order of process p is not always preserved in the execution time. Therefore, a change in the order of events on a process life line can lead to TP6. A subcategory of this type is known as race conditions. In race conditions, two or more processes compete in reaching a resource (sending a message to one process in our case).

For each type, we specify the situations that can lead to an implied scenario. Based on the required information for the occurrence of each type, different vectors are defined and extracted from the models for the detection algorithms.

5 EXPECTED OUTCOME

The classification of common types of emergent behaviors is based on the reasons and various conditions that can lead to each case. Therefore, the classification will help devising the detection algorithms, specifically to each category, which can also lead to suggest solutions on how to omit the problem or change the designs to fix the detected emergent behavior or implied scenario.

In this section, we focus on one of the emergent behavior types (TP4) that can occur because of asynchronous concatenation of MSCs. The asynchronous concatenation of MSCs is interpreted as concatenation of processes' actions independently. In other words, in an hMSC, one process can proceed to the next MSC, while another process is still involved in the previous MSC. The asynchronous concatenation of MSCs can cause two main problems that should be detected, in order to prevent emergent behaviors. These two problems, which we refer to as issues **I1** and **I2**, are investigated here. In general, for this type, the high level structure $\mathcal{G}_p$ of each process is identified. This structure is compared with the structure of hMSC. The $\mathcal{G}_p$ of each active process, should have the same loops, initial and final MSCs as the hMSC. If the initial MSC of $\mathcal{G}_p$ for an active process is not the same as initial MSC of hMSC, there is a chance to emerge an implied scenario. The reason is that, the active process can start to perform its actions without depending on the other processes. Thus, this process may start its tasks in its own initial MSC, while the other processes are not performing the same MSC, resulting in a new scenario that is not in the hMSC of the system.

5.1 Definitions

The basic definitions required for our modeling are described in this section.

5.1.1 Message Sequence Charts (MSC)

Let $P = \{ p, q, r, \ldots\}$ be the finite set of processes (components, agents) of the system, which are interacting with each other. We define $\sum_p$ as the set of communications that process p takes part in.

$$\textstyle\sum_p = \{p!\, q(m), p?\, q(m) | p, q \in P , m \in M\}$$

where M is the finite set of messages on the system. The $p!\, q(m)$ defines that process p sends message m to process q, and $p?\, q(m)$ defines that process p receives message m from process q. We define $\sum = \cup_{p\in P} \sum_p$.

Each MSC $\mathcal{M}$ shows a visual form of processes P and their interacting messages over the finite set of messages M. An MSC $\mathcal{M}$ is a structure $\mathcal{M} = (E, P, M, \mu, \leq, \alpha)$ where:

$E = \{e_1, e_2, \ldots\}$ is a finite set of $Send$ and $Receive$ events, where $S = \{s_1, s_2, \ldots\}$ is the set of $Send$ events and $R = \{r_1, rs_2, \ldots\}$ is the set of $Receive$ events.

P is a finite set of processes.

M is a finite set of messages.

$\mu: E \to \sum$ is a mapping function of events to messages and processes. For $p \in P$, $x \in \sum$ and $e \in E$:

$$E_p = \{e \mid \mu(e) \in \textstyle\sum_p, \mu(e) = x\}$$

The set of events on process $p \in P$ is:

$$E_p = \{e \mid \exists m \in M, e \in S, \mu(e) = p!\, q(m)\ or\ e \in R, \mu(e) = p?\, q(m)\}$$

And

$$E = \cup_{p\in P} E_p$$

$\leq$ is the set of total orders on E and μ.

$\alpha: S \to R$ maps the send events to receive events. For $e, e' \in E$ and $p, q \in P$, we define:

$$e <_{pq} e' \equiv \exists m \in M\ such\ that\ \mu(e) = p!\, q(m)\ and\ \mu(e') = q?\, p(m)$$

The $e <_{pq} e'$ relation explains that the message m sent by p at event e is received by q at event e'.

Each MSC $\mathcal{M}$ has a visual structure showing the set of processes P interacting to each other via sending or receiving messages. The process $p \in P$ in an MSC is shown by a vertical line representing the life line of the process. The interacting messages are shown with arrows (edges C) from one process to another process. The events E_p on process p have a ***local visual order*** represented by $<_p$, which is the total order of events on p as displayed in the MSC.

The ***visual order*** of an MSC $\mathcal{M}$ contains the local orders of all processes and the set of all edges C on that MSC:

$$(\cup_{p\in P} <_p)\ \cup C$$

5.1.2 High Level MSC

High level MSC (hMSC) is a structure $\mathcal{G} = (P, M, \boldsymbol{\mathcal{M}}, V, Ed, \complement, F_0, F_f)$, where P is the set of processes, M is the set of messages, $\boldsymbol{\mathcal{M}}$ represents the set of MSCs, V represents the vertices, $Ed \subseteq V \times V$ is the set of edges, and $\complement$ is a mapping function $\complement: V \to \boldsymbol{\mathcal{M}}$. The $F_0 \subseteq V$ and $F_f \subseteq V$ are the initial and final vertices of $\mathcal{G}$.

For each process $p \in P$ in $\mathcal{G}$, we define a structure $\mathcal{G}_p = (M_p, \boldsymbol{\mathcal{M}}_p, V_p, Ed_p, \complement_p, F_{0_p}, F_{f_p})$, where $\boldsymbol{\mathcal{M}}_p \subseteq \boldsymbol{\mathcal{M}}$ is the set of MSCs that p participates in, and has at least one action such that $\sum_p \neq \emptyset$. $M_p \subseteq M$ is the set of messages over $\boldsymbol{\mathcal{M}}_p$. V_p and $Ed_p \subseteq V_p \times V_p$ are the set of vertices and edges that are mapped by $\complement_p: V_p \to \boldsymbol{\mathcal{M}}_p$. The $F_{0_p} \subseteq V_p$ and $F_{f_p} \subseteq V_p$ are the initial and final vertices over V_p as visually displayed in $\mathcal{G}$.

Each process $p \in P$ follows a visual order on its events $E'_p = \cup_{\mathcal{M}_p} E_p$ as displayed in $\mathcal{G}_p$. We define $\sqsupset_p$ as ***global visual order*** of p over $\mathcal{G}_p$, which is the total order of events E'_p in $\boldsymbol{\mathcal{M}}_p$.

5.1.3 Example

Figure 1 illustrates the traffic situation, containing two highways HW1 and HW2, and two ramps, which are the exit ramps of HW2 and entrances of HW1. The flow of vehicles in the highways and ramps are depicted with dashed arrows. The inflow and outflow of the ramps to HW1 and from HW2 is controlled by ramp metering installations (RMI) RMI1 and RMI2. When congestion starts to build up on HW1 downstream of the second ramp, RMI2 will detect this and reduce the inflow. Likewise, when the exit ramp of HW2 starts to build up congestion, RMI1 and RMI2 will take action to reduce congestion. In order to have better performance and take better actions, RMI1 and RMI2 should be aware of future congestion. Therefore, the system is designed as a Multiagent System (MAS), which can be considered as a subset of DSS, in which the agents are capable of communicating to each other. In this way, they can react in advance and solve the congestion timelier.

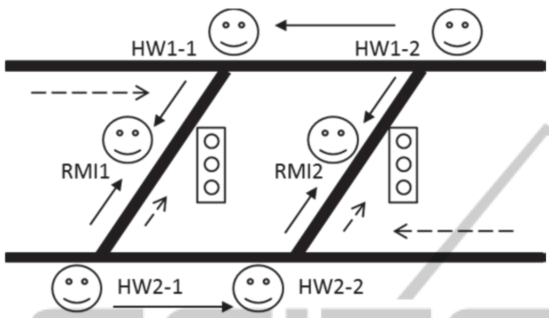

Figure 1: Multiagent System for congestion control.

The system has two highway agents for each highway and one RMI agent for each ramp. The highway agents have the capability of developing an image of the current traffic situation, depending on their own estimation and incoming messages from downstream agents. The arrows in Figure 1 indicate the possible communications of each agent. Highway agents can send “no problem”, “help-urgent high”, and “help-urgent low” messages to their neighbor agents and RMI agents. The RMI agents can take actions “make the traffic light green” or “make the traffic light red” based on the messages they receive from highway agents.

Two scenarios of this system are described using MSCs. These MSCs demonstrate a situation that the first agents of highway one and two, HW1-1 and HW2-1 respectively, requests urgent help from RMI1. In the first MSC M1, the request of HW2-1 is received sooner. Therefore, the RMI1 makes the traffic light green. This is shown in Figure 2.

In MSC M2, illustrated in Figure 3, HW1-1 sends its help request sooner and RMI1 agent makes the traffic light red.

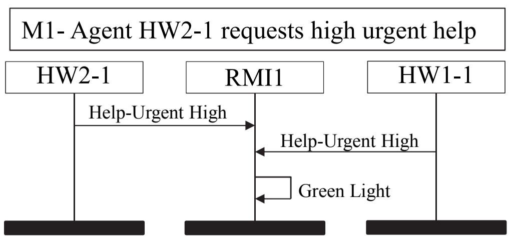

Figure 2: MSC M1- Agent in HW2 requests high urgent help.

The hMSC of these two MSCs is shown in Figure 4. The upside down triangle demonstrates the start and the regular triangle exhibits the termination of hMSC.

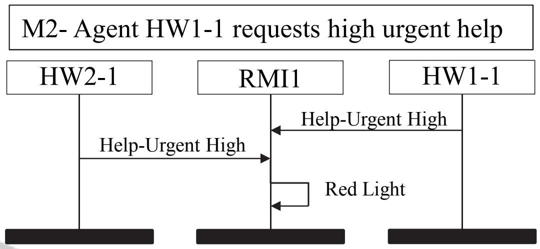

Figure 3: MSC M2- Agent in HW1 requests high urgent help.

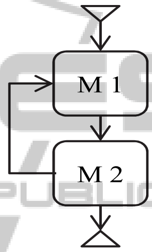

Figure 4: High level MSC of MAS congestion system.

5.2 Modeling Processes’ Interactions

We model the interaction of processes in each MSC_i into its corresponding interaction matrix IM_i. An interaction matrix is an square matrix of size $n = |P|$ equal to the total number of processes in the system. Two types of interaction matrices are defined that are used for various communication channels: *Send Interaction Matrix* and *Receive Interaction Matrix*.

We define a ***send vector*** $\delta_{p\mathcal{M}} = (a_1, a_2, \dots, a_n)$ for each process $p \in P$ in an MSC $\mathcal{M}$. The elements of $\delta_{p\mathcal{M}}$ can have one of the following values:

1) $if\ \exists q_i \in P, \exists s_j \in S, \exists e \in S\ and\ \exists m \in M\ such\ that\ \ \mu(e) = p!\,q_i(m) then\ a_i = s_j.$

2) $if\ \exists q_i \in P, for\ some\ s_j \in S, and\ for\ some\ e_k \in S\ and\ for\ some\ m_k \in M\ such\ that\ \ \mu(e_k) = p!\,q_i(m_k)\ \ and\ |\mu(e_k)| > 1\ then\ a_i\ is\ a\ set{:}\ a_i = \cup_j s_j.$

3) $if\ for\ q_i \in P, \nexists s \in S, \nexists e \in S, and\ \nexists m \in M\ such\ that\ \ \mu(e) = p!\,q_i(m) then\ a_i = \emptyset.$

The *send vector* δ_{pM} consists of n element where $n = |P|$. The $\delta_{p\mathcal{M}}$ is a set of sets $\{a_1, a_2, \dots, a_n\}$. The a_i is the set of all send events in which process p sends messages to another process $q_i \in P$ in an

MSC $\mathcal{M}$. The three cases above, represent that a_i can have exactly one member (case 1), more than one member (case 2), or be an empty set (case 3). $\delta_p = \cup_{\mathcal{M}} \delta_{p\mathcal{M}}$.

A ***Send Interaction Matrix*** SIM over P is a matrix that represents all communications in an MSC $\mathcal{M}$ that are of type "Sending". The entries of SIM are $\{sim_1, sim_2, ...\}$ where $sim_z \in S$ and row_p represents the p^{th} row in SIM and $row_p = \delta_{p\mathcal{M}}$.

We define a ***receive vector*** $\gamma_{p\mathcal{M}} = (b_1, b_2, ..., b_n)$ for each process $p \in P$ in an MSC $\mathcal{M}$. The elements of $\gamma_{p\mathcal{M}}$ can have one of the following values:

1) $if\ \exists q_i \in P, \exists r_j \in R, \exists e \in R\ and\ \exists m \in M\ such\ that\ \ \mu(e) = p?\,q_i(m) then\ b_i = r_j$.
2) *if ∃q_i∈P,for some r_j∈R,and for some e_k∈R and for some m_k∈M such that μ(e_k)=p?q_i (m_k) and |μ(e_k)|>1 then b_i is a set: b_i=∪_j r_j*
3) $if\ for\ q_i \in P, \nexists r \in R, \nexists e \in R, and\ \nexists m \in M\ such\ that\ \ \mu(e) = p?\,q_i(m) then\ b_i = \emptyset$.

The *receive vector* $\gamma_{p\mathcal{M}}$ consists of n elements where $n = |P|$. The $\gamma_{p\mathcal{M}}$ is a set of sets $\{b_1, b_2, ..., b_n\}$. The b_i is the set of all receive events in which process p receives messages from another process $q_i \in P$ in an MSC $\mathcal{M}$. The three cases above, represent that b_i can have exactly one member (case 1), more than one member (case 2), or be an empty set (case 3). $\gamma_p = \cup_{\mathcal{M}} \gamma_{p\mathcal{M}}$.

A ***Receive Interaction Matrix*** RIM over P is a matrix that represents all communications in an MSC $\mathcal{M}$ that are of type "Receiving". The entries of RIM are $\{rim_1, rim_2, ...\}\ where\ rim_z \in R$ and row_p represents the p^{th} row in RIM and $row_p = \gamma_{p\mathcal{M}}$. In FIFO communications, we have $RIM_i = (SIM_i)^T$ for their corresponding MSC $\mathcal{M}_i$, i.e. the two matrices SIM_i and RIM_i are transpose of each other. However, in other communication channels, SIM_i and RIM_i for their corresponding MSC $\mathcal{M}_i$, might be different based on the definitions of that channel, and $RIM_i \neq (SIM_i)^T$.

We define a ***state vector*** $\beta_{p\mathcal{M}} = (\cup(\delta_{p\mathcal{M}}, \gamma_{p\mathcal{M}}), <_p) = (\tau_1, \tau_2, ..., \tau_z)$ for each process $p \in P$ of an MSC $\mathcal{M}$ where $z = |E_p|$. The state vector $\beta_{p\mathcal{M}}$ is the total order set of E_p in the MSC $\mathcal{M}$ with respect to its local visual order $<_p$:

$$\tau_i = e_i \in E_p\ , \exists m \in M, such\ that\ \ e \in S, \mu(e) = p!\,q(m)$$

$$or\ e \in R, \mu(e) = p?\,q(m)$$

For process p we define $\beta_p = \cup_{\mathcal{M}} \beta_{p\mathcal{M}}$ with respect to its global visual order $\sqsupset_p$ over $\mathcal{G}_p$.

The ***state transition*** vector $\varphi_{p\mathcal{M}} = (\sigma_1, \sigma_2, ..., \sigma_x)$ for each process $p \in P$ of an MSC $\mathcal{M}$ where $x = |E_p|$, represents the senders of messages that cause a transition in states of process p in the MSC $\mathcal{M}$. Each action on the life time of process p in MSC $\mathcal{M}$ is either $p!\,q(m)$ or $p?\,q(m)$ for $m \in M$. For $m \in M$ and $\forall e \in E_p$:

$$\sigma_i = p\ if\ \mu(e) = p!\,q(m)\ or\ \sigma_i = q\ if\ \mu(e) = p?\,q(m)$$

We define $\varphi_p = \cup_{\mathcal{M}} \varphi_{p\mathcal{M}}$ as the total set of state transitions for process $p \in P$ over the set of $MSCs\ \boldsymbol{\mathcal{M}}$ in $\mathcal{G}_p$ with respect to its global visual order $\sqsupset_p$.

The ***basic state diagram*** $D_{p\mathcal{M}} = (N_{p\mathcal{M}}, \boldsymbol{E}_{p\mathcal{M}})$ is a directed graph that represents the visual structure of events E_p for each process $p \in P$ of an MSC $\mathcal{M}$, where $N_{p\mathcal{M}} = \{(st, snd) | st \in \beta_{p\mathcal{M}}\ and\ snd \in \varphi_{p\mathcal{M}}\}$ is the set of nodes and $\boldsymbol{E}_{p\mathcal{M}} = \{(sr, tg, m) | sr, tg \in N_{p\mathcal{M}}\ and\ m \in M\}$ is the set of edges, where $\exists e \in E_p, \sigma_i \in \varphi_{p\mathcal{M}}\ such\ that\ \mu(e) = p!\,q(m) or\ \ \mu(e) = p?\,q(m)$.

The ***state diagram*** $D_p = (N_p, \boldsymbol{E}_p)$ represents the total set of states of process $p \in P$ over the set of MSCs $\boldsymbol{\mathcal{M}}$. $D_p = \cup D_{p\mathcal{M}}$, where $N_p = \cup N_{p\mathcal{M}}$ and $\boldsymbol{E}_p = \cup \boldsymbol{E}_{p\mathcal{M}}$.

We define a ***minimum state diagram*** $D'_p = (N'_p, \boldsymbol{E}'_p)$ for process p, where $N'_p \subseteq N_p$ and $\boldsymbol{E}'_p \subseteq \boldsymbol{E}_p$. For $n' = (st, snd)$, $n' \in N'_p, st \in \beta_p$, $snd \in \varphi_p$, $q \in P, n_1, n_2 \in N_p$, and $e \in E'_p$:

$if\ st_1 = st_2\ and\ snd_1 = snd_2\ then\ n_1 = \ n_2 = n'$.

For $edg' = (sr, tg, m)$, $edg' \in \boldsymbol{E}'_p, sr, tg \in N_p$, $sr = n_1 = \ n_2 = n'$, $m, m_1, m_2 \in M$, $edg_1, edg_2 \in \boldsymbol{E}_p$, $edg_1 = (sr, tg_1, m_1)$, and $edg_2 = (sr, tg_2, m_2)$:

1) if $tg_1 = tg_2\ and\ m_1 = m_2\ then\ edg_1 = edg_2 = edg'$.
2) if $tg_1 = tg_2\ and\ m_1 \neq m_2\ then\ edg_1 \neq edg_2$ and we define two edged $edg'_1 = (sr, tg_1, m_1)$ and $edg'_2 = (sr, tg_2, m_2)$.
3) if $tg_1 \neq tg_2\ and\ m_1 = m_2\ then\ edg_1 \neq edg_2$ and we define two edged $edg'_1 = (sr, tg_1, m_1)$ and $edg'_2 = (sr, tg_2, m_2)$.

In the minimum state diagram for each process, we merge the vertices based on the states and the state transition of that state. Therefore, if two vertices in has the same vertex and edge, they are merged in one vertex in D'_p (case 1). However, if any of the states or state transitions (defined in the edges) are different, two edges are displayed in the minimum state diagram D'_p. This indicates that the senders of messages that change a state transition are considered in merging the vertices. The latter illustrates a branch in D'_p (case 2 and 3).

Note that since the events E_p on process p have a *local visual order* $<_p$, the $<_p$ relation is preserved in the send, receive, and state vectors δ_{pM}, γ_{pM}, and β_{pM} in the MSC $\mathcal{M}$. For the same reason, δ_p, γ_p, and β_p follow the global visual order $\sqsupset_p$ in its high level structure $\mathcal{G}_p$.

5.3 Emergent Behaviors of Asynchronous Concatenation

Emergent behaviors, also known as implied scenarios, are referred to as unexpected behaviors of processes (agents) during their execution, which were not seen in their design. The word emergent behavior is mostly used for the unexpected behavior of one process without considering other processes in the system. Implied scenarios are used for a situation that an unexpected behavior occurs when considering all processes in the system. In this case, a new scenario is implied, and it is considered as system level emergent behavior. Emergent behaviors and implied scenarios both refer to unexpected behaviors in the system; the only difference is that the unexpected behavior is examined in component or system level. Therefore, we may use these terms interchangeably in this paper.

In asynchronous concatenation of MSCs in a HMSC, the processes may proceed in next MSCs in different times (Muscholl and Peled, 2005). Let $\mathcal{M}_1, \mathcal{M}_2 \in \boldsymbol{\mathcal{M}}$ be two MSCs from the set $\boldsymbol{\mathcal{M}}$ that contains all MSCs of the system, and $\mathcal{M}_1$ precedes $\mathcal{M}_2$ in the HMSC $\mathcal{G}$. Consider processes $p, q \in P_{\boldsymbol{\mathcal{M}}}$ where $P_{\boldsymbol{\mathcal{M}}}$ is the finite set of processes (agents) of $\boldsymbol{\mathcal{M}}$; and p and q have some actions in both $\mathcal{M}_1$ and $\mathcal{M}_2$. In asynchronous concatenation of MSCs, processes do not wait until all other processes accomplish their actions in one MSC. Therefore, while process p is still involved in $\mathcal{M}_1$, process q may proceed to perform its actions in $\mathcal{M}_2$ that we refer to as issue **I1**.

Consider the MAS system in the example. The agent HW2-1 in MSC M1 can proceed to M2 and send a request of "high urgent help" to agent RMI1; while, HW1-1 is still proceeding its actions in M1. This can result in a repetition in executing MSC M1. Thus, an emergent behavior that is shown in Figure 5 (a) can happen. The same situation can occur for HW1-1 in M2. Consequently, the MSC M2 is repeated and emergent behavior of Figure 5 (b) can arise.

The asynchronous concatenation of MSCs may result in another issue **I2**. Let a third process $r \in P_{\boldsymbol{\mathcal{M}}}$ that is involved in $\mathcal{M}_1, \mathcal{M}_2 \in \boldsymbol{\mathcal{M}}$ where it has interactions with both processes p and q in $\mathcal{M}_1$ and $\mathcal{M}_2$ and it receives some messages from both processes p and q.

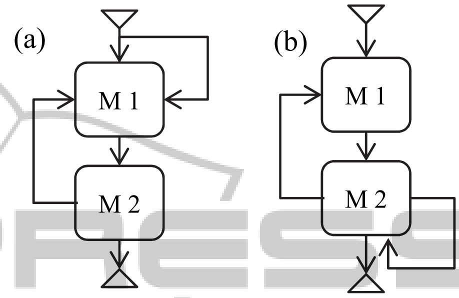

Figure 5: Two emergent behaviors caused by issue I1.

In this case, there is no control over the receive events on process line of r when its senders are different. This is referred to as a *race condition* in the literature. The same situation can occur when $p, q, r \in P_{\boldsymbol{\mathcal{M}}}$ and process r receives some messages from process p and q in MSC $\mathcal{M} \in \boldsymbol{\mathcal{M}}$.

Let add another MSC to the MAS example that is depicted in Figure 6. In this MSC, M3, agent HW2-1 send a low urgent help request to RMI1. Therefore, RMI1 asks HW1-1 for its situation to take the best action. Since the urgency of HW1-1 is high, RMI1 makes the traffic light red.

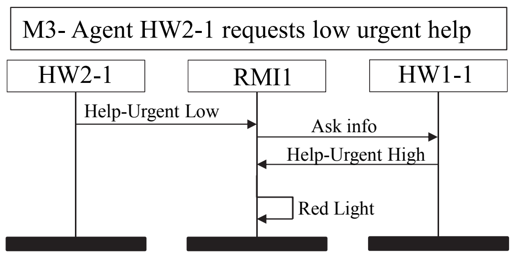

Figure 6: MSC M3- Agent in HW2 requests low urgent help.

An implied scenario that can occur is shown in Figure 7. The agent HW2-1 can proceed to MSC M1 and send a high urgent request. Since there is no control on the receiving messages of RMI1 that are sent by different agents (HW2-1 and HW1-1), RMI1 may proceed to M1 and make the traffic lights green.

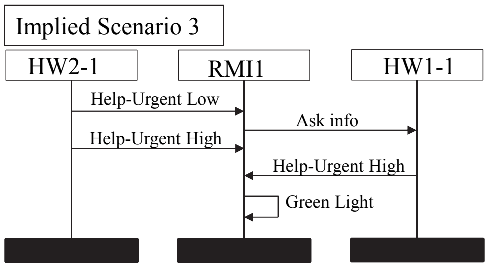

Figure 7: Implied scenario caused by issue I2.

5.3.1 Define Detection Rules for Issue I1

We define active and passive processes that are processes that, whether or not, have control to start an MSC, respectively. Since an active process sends its first message in an MSC, the process does not need to wait for an action that changes its state. Therefore, an active process is the sender of its first action that leads to a state transition for its first state. However, this is not held for a passive process. A passive process should wait for the sender of its action to send it a message and changes its state. Therefore, for a passive process, starting an MSC is not in its control and depends on other processes. The definitions for active and passive processes are:

Let processes $p, q \in P$ have some actions in an $MSC\ \mathcal{M} \subseteq \boldsymbol{\mathcal{M}}$. We define process p as an ***active process*** in $\mathcal{M}$ if for the first action of p in its local visual order $<_p$, the following condition is satisfied in $\mathcal{M}$:

$$\exists m \in M, e \in S |\ \mu(e) = p!\,q(m)$$

Let processes $p, q \in P$ have some actions in an $MSC\ \mathcal{M} \subseteq \boldsymbol{\mathcal{M}}$. We define process p as a ***passive process*** in $\mathcal{M}$ if for the first action of p in its local visual order $<_p$, the following condition is satisfied in $\mathcal{M}$:

$$\exists m \in M, e \in S |\ \mu(e) = p?\,q(m)$$

Let $\boldsymbol{\mathcal{M}}_{pa} \subseteq \boldsymbol{\mathcal{M}}$ as the set of MSCs in which a process $p \in P$ is an active process.

Rule R1: An active process $p \in P$ can lead to issue **I1** in all MSCs $\mathcal{M} \subseteq \boldsymbol{\mathcal{M}}_{pa}$ if there is no timing or control over the MSCs. The reason is that process p has the control of starting an action in a new MSC in $\boldsymbol{\mathcal{M}}_{pa}$.

Rule R2: Let process $p \in P$ in hMSC $\mathcal{G} = (P, M, \boldsymbol{\mathcal{M}}, V, Ed, \complement, F_0, F_f)$, that has $\mathcal{G}_p$ structure $\mathcal{G}_p = (M_p, \boldsymbol{\mathcal{M}}_p, V_p, Ed_p, \complement_p, F_{0_p}, F_{f_p})$, be an active process. Process p can lead to issue **I1** if one of the following conditions holds:

1) $F_{0_p} \neq F_0\ and\ F_{0_p} \subseteq \boldsymbol{\mathcal{M}}_{pa}$
2) $F_{0_p} = F_0\ and\ F_{f_p} \neq F_f\ and\ \exists v \in V_p\ such\ that\ \complement_p(v)\ \subseteq \boldsymbol{\mathcal{M}}_{pa}$
3) $F_{0_p} \neq F_0\ and\ F_{f_p} \neq F_f\ and\ \exists v \in V_p\ such\ that\ \complement_p(v)\ \subseteq \boldsymbol{\mathcal{M}}_{pa}$

When the $\mathcal{G}_p$, the structure of MSCs that an active process $p \in P$ follows, is different from the structure of hMSC $\mathcal{G}$, which is defined as a global view for all processes, it might lead to issue **I1**. The first condition defines a situation in which the initial MSC of $\mathcal{G}_p$ is different from the initial MSC in $\mathcal{G}$. Therefore, process p can start its high level structure, without following the scenarios that the whole system performs. The second condition, states that the initial MSCs in $\mathcal{G}_p$ and $\mathcal{G}$ are the same. However, the termination MSC for p is different. In this situation, the active process p starts its initial MSC when the other processes start performing their scenarios. However, it does not follow the same scenarios that other processes are performing in other iterations of the hMSC. Because the termination nodes (final vertices) are different, p can start performing its high level structure $\mathcal{G}_p$ again, while the other processes are performing actions in the rest of MSCs in $\mathcal{G}$. The third condition, is a more general condition in which process p has different initial and termination MSCs in $\mathcal{G}_p$ with initial and termination MSCs in $\mathcal{G}$, respectively.

Rule R3: Issue **I1** can occur when process p has a loop in its $\mathcal{G}_p$ structure, which is an internal loop in hMSC $\mathcal{G} = (P, M, \boldsymbol{\mathcal{M}}, V, Ed, \complement, F_0, F_f)$. The loop in $\mathcal{G}$ is an internal loop when it does not include the initial and termination vertices F_0 and F_f. Process p can be either an active or a passive process.

When p performs actions in an internal loop, it may execute the loop more than other processes. The termination MSCs are different in $\mathcal{G}_p$ and $\mathcal{G}$. Therefore, p can start performing $\mathcal{G}_p$ again, while the other processes are continuing to perform their actions in other MSCs in $\mathcal{G}$.

Rule R4: Consider an active or passive process p that has a loop in $\mathcal{G}_p = (M_p, \boldsymbol{\mathcal{M}}_p, V_p, Ed_p, \complement_p, F_{0_p}, F_{f_p})$ from F_{f_p} to F_{0_p}. If this loop is an internal loop in hMSC $\mathcal{G} = (P, M, \boldsymbol{\mathcal{M}}, V, Ed, \complement, F_0, F_f)$, process p can cause issue **I1**.

The reason is similar to the reason of rule **R3**. The final MSCs in $\mathcal{G}_p$ and $\mathcal{G}$ are different. Therefore, process p can start executing its MSCs, while the other processes are continuing their actions in other MSCs.

5.3.2 Define Detection Rules for Issue I2

Suppose that processes $p, q, r \in P$ have interactions with each other in an MSC $\mathcal{M}$. If process p has at least one receive message from each of the other processes q and r, and there is no control over the order of receive messages for process p with respect to its visual order $<_p$, a race condition can occur. In this situation the visual order $<_p$ in MSC $\mathcal{M}$ is not preserved.

Rule R5: Let $\gamma_{p\mathcal{M}} = (b_1, \dots, b_r, \dots, b_q, \dots, b_n)$ be the receive vector of process p in an MSC $\mathcal{M}$. The elements b_r and b_q represent the set of events that p receives from processes r and q respectively. Let $\gamma_{p\mathcal{M}}' = (b_1, \dots, b_q, \dots, b_r, \dots, b_n)$ be the new receive vector for process p, where the order of b_r and b_q has changed, compared to $\gamma_{p\mathcal{M}}$. We have $\beta_{p\mathcal{M}} = (\cup\,(\delta_{p\mathcal{M}}, \gamma_{p\mathcal{M}}), <_p)$ as the state vector of p and $\beta_{p\mathcal{M}}' = (\cup\,(\delta_{p\mathcal{M}}, \gamma_{p\mathcal{M}}'), <_p)$ as the new state vector of process p.

If $\beta_{p\mathcal{M}}'$ is not in the set of $\beta_p = \cup_{\mathcal{M}}\,(\beta_{p\mathcal{M}})$, then process p can cause issue **I2** because of a change in its receive orders:

$$
\begin{aligned}
if\ \exists m, m' \in M;\ e, e' \in E_p; \mu(e), \mu(e') \\
\in \textstyle\sum_p\ such\ that\ \mu(e) \\
= p?q(m)\ and\ \mu(e') \\
= p?r(m')\ and\ \neg\left(\beta_{p\mathcal{M}}' \subseteq \beta_p\right)
\end{aligned}
$$

$then\ \beta_{p\mathcal{M}}'$ is an emergent behavior X for p.

Rule **R5** indicates that a change in the order of receiving messages for process p should be in the events of process p in other MSCs, or receiving of such messages should be controlled, in order to prevent an emergent behavior.

Rule R6: Let $\gamma_{p\mathcal{M}}' = (b_1, \dots, b_q, \dots, b_r, \dots, b_n)$ be a new receive vector for process p, where the order of b_r and b_q has changed, compared to $\gamma_{p\mathcal{M}}$ and $\exists \mathcal{M}' \in \boldsymbol{\mathcal{M}}$ such that $\beta_{p\mathcal{M}}' \subseteq \beta_p$. Let $D_{p\mathcal{M}} = (N_{p\mathcal{M}}, \boldsymbol{E}_{p\mathcal{M}})$ be the associated basic state diagram for receive vector $\gamma_{p\mathcal{M}}$ and state vector $\beta_{p\mathcal{M}}$ in $\mathcal{M}$. Consider $D_{p\mathcal{M}}' = (N_{p\mathcal{M}}', \boldsymbol{E}_{p\mathcal{M}}')$ as the associated basic state diagram for receive vector $\gamma_{p\mathcal{M}}'$ and state vector $\beta_{p\mathcal{M}}'$ in $\mathcal{M}'$. We define $n_q', n_r' \in N_{p\mathcal{M}}'$ as the corresponding nodes in $D_{p\mathcal{M}}'$, for the states of process p that receive messages from process q and r, respectively. If $D_{p\mathcal{M}}'$ is not in $D_p = \cup\, D_{p\mathcal{M}}$, then $\beta_{p\mathcal{M}}' = (\cup\,(\delta_{p\mathcal{M}}, \gamma_{p\mathcal{M}}'), <_p)$ is an emergent behavior X.

Rule **R6** defines more restrictive criteria that a process can have a behavior that leads to issue **I2**. Other than the new receive vector in the states of process p, the state transition vectors should be considered as well; to check whether or not a process can lead to **I2** and show an emergent behavior.

A change in the order of states b_1 and b_2, resulted from communications with processes q and r, in the receive states of process p, creates a new receive vector $\gamma_{p\mathcal{M}}'$. The $\gamma_{p\mathcal{M}}'$ can exist in the state vectors of p in another MSC $\mathcal{M}'$. However, the communicating processes associated to states b_1 and b_2 in $\mathcal{M}'$, may be different, namely, processes y and z. Therefore, the two state vectors $\beta_{p\mathcal{M}}$ and $\beta_{p\mathcal{M}}'$, are not considered as similar state vectors when considering their associated state transitions; because their communicating processes for these two states (b_1 and b_2) are different. In this situation, although the state vectors in the two MSCs are the same, process p can still show an emergent behavior, because of a difference in its state transition vectors.

6 STAGE OF THE RESEARCH

Until now, the categories of emergent behaviors are formalized. For each category, various conditions and reasons that can lead to an implied scenario are investigated, and some algorithms are devised. The first parts of the methodology, including transforming the designs into interaction matrices, and detecting components with no emergent behaviors are implemented. The current research trend is focused on testing algorithms on large scale systems, using communication of processes in Linux clusters as the test bed.

ACKNOWLEDGEMENTS

This research is supported by a grant from Izaak Walton Killam Memorial Scholarship, Alberta Innovates Technology Futures and partially from Natural Sciences and Engineering Research Council of Canada.

REFERENCES

Aggarwal, C. C. (2011). An introduction to social network data analytics (pp. 1-15). Springer US.

Alur, R., Etessami, K., & Yannakakis, M. (2005). Realizability and verification of MSC graphs. Theoretical Computer Science, 331(1), 97-114.

Bhateja, P., Gastin, P., Mukund, M., & Kumar, K. N. (2007, January). Local testing of message sequence charts is difficult. In Fundamentals of Computation Theory (pp. 76-87). Springer Berlin Heidelberg.

Briand, L. C. (2010). Software Verification—A Scalable, Model-Driven, Empirically Grounded Approach. In Simula Research Laboratory (pp. 415-442). Springer Berlin Heidelberg.

Chaki, S., Clarke, E., Grumberg, O., Ouaknine, J., Sharygina, N., Touili, T., & Veith, H. (2005, January). State/event software verification for branching-time specifications. In Integrated Formal Methods (pp. 53-69). Springer Berlin Heidelberg.

Chakraborty, J., D'Souza, D., & Kumar, K. N. (2010). Analysing message sequence graph specifications. In Leveraging Applications of Formal Methods, Verification, and Validation (pp. 549-563). Springer Berlin Heidelberg.

Fard, F. H., & Far, B. H. (2013, August). Detecting distributed software components that will not cause emergent behavior in asynchronous communication style. In Information Reuse and Integration (IRI), 2013 IEEE 14th International Conference on (pp. 201-208). IEEE.

Holzmann, G. J., & Smith, M. H. (2002). An automated verification method for distributed systems software based on model extraction. Software Engineering, IEEE Transactions on, 28(4), 364-377.

Iglesias, A. (2009). Software Verification and Validation of Graphical Web Services in Digital 3D Worlds. In Communication and Networking (pp. 293-300). Springer Berlin Heidelberg.

ITU-T UNION, SERIES Z: languages and general software aspects for telecommunication systems - Formal description techniques (FDT) – Message Sequence Chart, 2004, ITU-T Recommendation Z.120. p. 136.

Krüger, I. H. (2000). Distributed system design with message sequence charts(Doctoral dissertation, Technische Universität München, Universitätsbibliothek).

Letier, E., Kramer, J., Magee, J., & Uchitel, S. (2005, May). Monitoring and control in scenario-based requirements analysis. In Proceedings of the 27th international conference on Software engineering (pp. 382-391). ACM.

Mousavi, A. (2009). Inference of emergent behaviours of scenario-based specifications (Doctoral dissertation, UNIVERSITY OF CALGARY).

Muscholl, A., & Peled, D. (2005). Deciding properties of message sequence charts. In Scenarios: Models, Transformations and Tools (pp. 43-65). Springer Berlin Heidelberg.

Song, I. G., Jeon, S. U., Han, A. R., & Bae, D. H. (2011). An approach to identifying causes of implied scenarios using unenforceable orders. Information and Software Technology, 53(6), 666-681.

Uchitel, S. (2003). Incremental elaboration of scenario-based specifications and behaviour models using implied scenarios (Doctoral dissertation, Imperial College London (University of London)).

Uchitel, S. (2009). Partial Behaviour Modelling: Foundations for Incremental and Iterative Model-Based Software Engineering. In Formal Methods: Foundations and Applications (pp. 17-22). Springer Berlin Heidelberg.

Uchitel, S., Kramer, J., & Magee, J. (2002, November). Negative scenarios for implied scenario elicitation. In Proceedings of the 10th ACM SIGSOFT symposium on Foundations of software engineering (pp. 109-118). ACM.