# On the Usage of Network Visualization for Multiagent System Verification

**Fatemeh Hendijani Fard and Behrouz H. Far**

**Abstract** Multiagent Systems (MAS) consists of many software agents that interact to each other to perform their actions and achieve system goals. Due to the growing demand of Distributed Software Systems (DSS) and MAS as a branch of DSS, the verification of these systems has taken a special attention. The verification of these systems is required because MAS and DSS are large scale systems, where a failure can result in a huge amount of cost or damage. One major field is verification of MAS designs to prevent cost of fixing problems after implementation and deployment. The agents and interactions among them form a network of agents. This network can be used for verification of MAS from different perspectives and by various techniques. Visualizing agents' networks can lead to detect a special type of unexpected behavior in MAS referred to as emergent behaviors and implied scenarios. These unexpected behaviors are more probable in large scale systems because the functionality and control are distributed and there is lack of central controller in MAS and DSS. Consequently, a new scenario can be implied to the system at run time which may not be an acceptable behavior in the system. In this article, visualization techniques are applied to form three different networks extracted from the designs of MAS for this purpose. These networks are used to detect emergent behaviors and implied scenarios in the system. The methodology, system architecture, data preparation and visualization, required network definitions, and results are verified through case studies.

**Keywords** Network visualization technique · Multiagent systems verification · Scenario-based software engineering · Emergent behaviors · Implied scenario

F. Hendijani Fard · B.H. Far (✉)
Department of Electrical and Computer Engineering, University of Calgary, Calgary, AB, Canada
e-mail: far@ucalgary.ca

F. Hendijani Fard
e-mail: fhendija@ucalgary.ca



J. Kawash (ed.), *Online Social Media Analysis and Visualization*,
Lecture Notes in Social Networks, DOI 10.1007/978-3-319-13590-8_10

## 1 Introduction

In recent years, industrial applications of agent based systems have gained popularity due to certain characteristics of agents, such as autonomous, collaborative, and proactive characteristics [1]. Other characteristics, such as information hiding, ability to learn, knowledgeability, autonomy, and mobility, are among features that have made them ideal for use in Distribute Software Systems (DSS) [2]. One of the problems that arise in DSS and Multi Agent Systems (MAS), is the lack of a central controller that may cause the components/agents a new behavior to emerge at run time, which was not seen in the system specifications. This unexpected behavior, which is known as emergent behavior at the *component level* (i.e. considering behavior of agents individually) and implied scenario at the *system level* (i.e. considering the system behavior), may cause critical damage [3]. Since these are unexpected behaviors, which were not seen in the designs, they should be detected and verified before system deployment. The acceptance or denying these behaviors is what the software designer and stakeholders should decide on. However, they should be aware of such behaviors earlier to consider these behaviors in the design or change the designs to avoid them. The early detection of these behaviors reduces the cost and time for fixing them [4]. It is stated that "*Detecting the causes of faults early may reduce their resulting costs by a factor of 100 or more*" [5].

There are many approaches for the detection of emergent behaviors in the requirements and design phases of scenario based software systems. In these systems, the agents/components' behaviors and their interactions are shown through graphical elements such as Messages Sequence Charts (MSC) or UML Sequence Diagrams. In this paper, the MSCs are used as inputs of the system since they are widely used for DSS. Figure 1 is an example of presenting system scenarios in the form of MSCs. These are parts of the scenarios related to a multiagent system for managing a greenhouse. The agent $A_w$ is the agent responsible for irrigation of plants. This agent interacts with other agents, such as $A_t$ and $A_m$ (agents responsible for heat balancing and providing minerals) to perform effectively. Suppose that the only acceptable interactions among agents of this system are shown in Fig. 1.

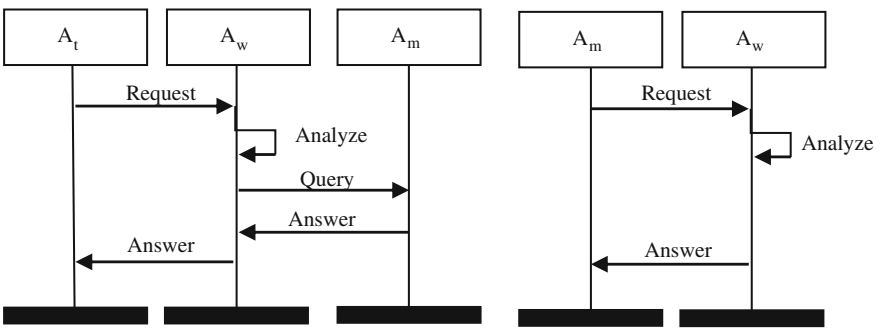


**Fig. 1** Two scenarios showing part of agents' interactions for Greenhouse MAS

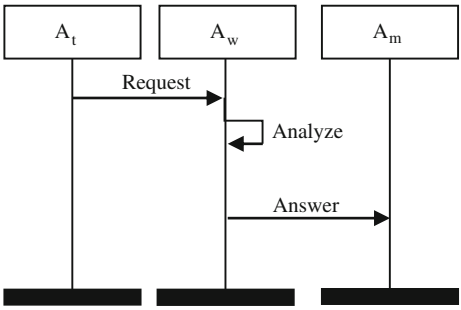


**Fig. 2** An emergent behavior in Greenhouse MAS

The acceptable actions of agent $A_w$ are shown in the MSCs of Fig. 1; however, this agent may behave differently as shown in Fig. 2. Therefore, if a new scenario, such as Fig. 2 happens, it can lead to a problem in the system.

Figure 2 is an example of an emergent behavior. From the point of view of agent $A_w$, this agent has a choice to follow, when it receives the "Analyze" message and reaches its third state. Therefore, it neither follows the first MSC nor the second one. This behavior is called an emergent behavior of $A_w$. More explanations as to the reason and the effects of this example are presented in next sections. It is worth mentioning that the decision of accepting or declining implied scenarios and emergent behaviors depends on the stakeholders of the system, but they should be detected and defined earlier to redesign the system to handle these situations.

Most research in the detection of emergent behaviors and implied scenarios use model checking approaches that try to verify whether the model explaining the behavior of components/agents satisfies the requirements and designs. These approaches use various behavioral modeling techniques and finite state machines to be verified against the extracted languages [6–8]. One of the issues with model checking approaches other than the problem of state space explosion (when they face large scale systems) [9–16], is that most of them cannot find the exact point or cause of emergent behavior and just identify the existence of an emergent behavior in the system [17]. Therefore, this is a flaw that these approaches do not suggest solutions to the designer and do not fix the problem either by the designer or automatically.

In this paper, visualization of the agents' networks is taken into account as a major tool for emergent behavior and implied scenario detection. Three different networks are extracted from MSCs and visualized to detect the points in which an unexpected behavior can happen. It can both demonstrate where and why the unexpected behavior can occur and provide the reasons for and the effects of emergent behaviors. One of the networks is associated with events on any individual agent for component level analysis and the other two networks illustrate the agents' interactions with each other and how they see the sequence of actions that should be performed at the system level. These networks are used to show violations to the designers and stakeholders and to verify the behavior of agents. Although, in this paper, the complexity of existing approaches and the present work is not the main issue, in our technique, model checking approaches are avoided since they require behavioral modeling or synthesis

phase, which is considered to be even more complex than model checking [18]. To avoid synthesis, we have devised a new technique for emergent behavior and implied scenario investigation [19, 20] using interaction matrices. The scalability issues of this technique is explained more in [21]. In this paper, the interaction matrices are used to extract and demonstrate the three mentioned networks: agent's states network, agents' interaction network, and graph of sequence of MSCs for each agent. Based on our knowledge, this visualization for MAS verification against emergent behavior and implied scenarios is among contributions of this research.

It is worth mentioning that these networks are semantically and visually different from the state machines in Finite Automata and formal methods. This is more explained in the Discussion part of this paper. The related works and basic information about scenario based systems and MSCs, emergent behavior and its detection approaches, and applications of social network analysis on MAS verification are presented in next section. It is followed by the methodology which explains system architecture, general steps of methodology, component and system level analysis, data preparation, data visualization, verifying the networks, and how to generate reports in Sect. 3. Two examples are presented to illustrate the methodology and results verification. The first example is a MAS greenhouse, which is used to explain component level analysis. The second example is from Uchitel's et al. [22] and is used to explain the system level analysis. In the rest of this paper the words *component* and *agent* may be used interchangeably and by both words we mean the word *software agent*.

## 2 Related Works

### *2.1 Scenario Based Systems and Message Sequence Charts*

In software engineering, a practical approach is describing software requirements using scenarios. Scenarios are stories about people and interactions, including agents and actors and sequences of actions and event. Scenarios can describe different levels and details with multiple perspectives. For different purposes, scenarios can provide abstraction for the designer and problem solving [23]. The visual forms of scenario based specifications like UML Sequence Diagrams (SD) and Message Sequence Chart (MSC) can show the software behavior. Software system components (processes) or inter-object and inter-processes are shown using vertical lines with their interaction messages and send/receive events respectively in MSC or SD [24, 25]. These specifications can be used for model checking, formal verification, or monitoring for emergent behavior detection [26].

The MSCs are visual and formal description techniques for software requirements and are widely used for MAS and DSS [27–30]. DSS has gained a broad attention in recent years for developing various systems and a main process in DSS is how

to show the system components and their cooperation [29]. This process in many works is done using MSCs. Development of MSC, its definitions and semantics has been standardized and published by Telecommunication Standardization Sector of International Telecommunication Union (ITU) [31]. MSC describes the communication behavior of system components and their environment in the form of graphical representation and message interchanging.

In an MSC each vertical line shows a component and the messages between them are shown with arrows. The name of the component is written above its line. Each message is shown with the label above it. The related scenario shows the MSC name. Figure 1 shows an example of a MSC.

A formal description of MSC from [32] is defined as a set (E, L, C, $\mu$, $<$, M) where:

- E is the set of events consists of send and receive ones.
- L is a finite set of labels
- C is the set of components
- $\mu$ is the mapping of events to labels and components
- $<$ is the set of total orders on the events and $\mu$
- M maps the send events to receive ones

In an MSC, the messages on the life line of each component are referred to as events on that line. These events can change the state of a component. The network of these states for each component is referred to as agent's state in the rest of this paper. Going from one state to another state or changing the state for an agent occurs by receiving or sending a message to that agent. The message (event) that changes the state of an agent is referred to as state transition in next sections.

The graphical means for showing how a set of MSCs can be combined are shown either with Message Sequence Graph (MSG) or high level message sequence charts (hMSC). These two are structured models of MSC. MSG has operations such as choice and repetition. MSG has MSCs as its nodes and the edges represent the concatenation of MSCs. A choice or branching in MSG can be shown with a hexagonal. An hMSC is a graph like formation and each node can be either an MSC or an MSG. In hMSC, the start node is shown with an upside down triangle and the end MSCs is shown with a triangle. In many researches MSG and hMSC are considered almost the same. MSC, hMSC, and MSG can be considered as the early models of the system [33].

## *2.2 Emergent Behavior*

Emergent behavior is defined as the unexpected behaviors of software components in the execution time, while it was not seen in their requirement and design specification [4]. These behaviors may cause irreparable damages to the system and therefore detecting them in early phases of software lifecycle is valuable [6, 34]. Emergent behaviors are also 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 or unexpected behavior. This happens because the model is not the precise equivalent of the requirements specification [17, 35, 36].

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 [37–39]. This state in some researches is referred to as identical states [4].

The problem of detecting emergent behavior can be classified in two categories which have different names based on the phase of software lifecycle and the scope they are used. Component level emergent behavior detection investigates the unexpected behaviors in component level i.e. considering the individual component and not its interaction with other components of the software. This refers to component fault in some researches where the behavior of the implemented component does not satisfy its specified behavior [5]. The second class is the detection of emergent behavior in system level i.e. investigating the components and their interactions as a system. Because even if the components have no emergent behavior, they can behave differently when they are integrated and have interaction with other components of the system. This is referred to as emerging an implied scenario in the system. The reason of having implied scenarios is explained through an example in next sections.

## *2.3 Emergent Behavior Detection Approaches*

The existing approaches consider detection of emergent behavior, implied scenarios, or both. One approach for model checking in the requirement is Alur et al. methodology [33]. His works define a detailed explanation of model checking of Message Sequence Chart (MSC), Message Sequence Graph (MSG), and high-level Message Sequence Chart (hMSC) [6]. All of the linearization of this model (MSC and hMSC) is then checked against an automaton which is defined by the message alphabet $\sum$, words, languages, states, and transitions which show the behavior of each process/component. If the intersection of the automaton (showing unwanted behavior of the system) and the linearization is empty, it shows that the defined model with MSC satisfies the requirements. He also defines the realizability of MSC and MSG which means the implementation should generate the exact behaviors specified in the graph. The safe realizability has polynomial-time solution for MSC [40]. One problem with model checking with Finite State Machine (FSM) is the required processes to prepare the model: Flattening, remove cycles, etc. [41].

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 with algorithms in different steps 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). In this translation, they detect

conflicts and indentical states in merging the scenarios to have justified merging of scenarios [30, 42]. Their most work is on the design part or synthesizing scenarios to state machines and code [43, 44].

Ben-Abdallah explains the non-local branching choice which causes the emergent behavior in system. This problem may arise from the abstraction view of MSC and not having semantics or specific interpretation of the implemented process when MSCs face a branching choice in hMSC [45, 46].

Muccini inspects the effects of non-local branching choice on emerging implied scenarios and explains the reason of an implied scenario generation. When the state machines are synthesized from scenarios (which explain a model and specification of the system), a set of behaviors are presented by state machines which was not in the scenarios [35]. 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 as the specification language. 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. Their methodology considers both synchronous and asynchronous communications [17, 47].

Uchitel et al. provide a methodology and tool for detection of implied scenarios. Their algorithm builds Labeled Transition Systems (LTS) for behavioral modeling of MSC and hMSC (as semantics of MSC) with synchronous communication. They have implemented their work in a tool named Labelled Transition System Analyzer (LTSA) [8, 22, 37, 48, 49]. The work is later extended by Letier et al. to detect input-output implied scenarios [50]. They also refer to the rejected implied scenarios by the stakeholders as negative scenarios and define behavioral constraints with LTS for eliciting them from implied scenarios. Kramer et al. use Timed automata and the behavior model specified by LTS to animate the behavior models in LTSA. The Timed automata adds local clocks to LTS [51].

Kumar et al. discuss the problems with main researches in this field. He discusses that many researches are not implementable and amendable, or do not show correctness. They use 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 [7]. Correspondingly, they discuss that "without message labels, if we observe one process at a time, checking for implied scenarios is decidable" [39]. There are other works presented in [52, 53] regarding the theories of message passing automata and regular MSC.

This paper uses a different approach for verifying MSC specifications using interaction matrices. Several pieces of information can be extracted from interaction matrices: number of messages (events) for each component/agent in each MSC, type of messages (send or receive), the order of messages in the MSC, the interacting agents, the senders and receivers of messages, and the states of each agent.

Using these matrices helps to visualize specific networks required for detection of emergent behaviors and implied scenarios. The network demonstration also helps to detect where the emergent behavior has happened (in terms of the events of one agent in the system), which possible MSCs it affects, and probably the origin of such a behavior. This ability will help the designer to fix the problem rather than just notifying of the existence of emergent behavior.

## *2.4 Social Network Analysis in MAS Communication Verification*

Social networks as a basic view are a series of individuals and their interaction such as Facebook. However, many other forms of social networks exist. Social networks can be considered as various types of interactions in different communities [54, 55]. The entities are considered as nodes with edges representing links between them which can be shown in matrices [56]. Many analyses have been proposed to social networks to specify different characters and in various applications [57, 58]. Some measurements in SNA are link or node measures like tie strength, betweenness, degree, authority, and hub measure, etc. [59] which are useful in many aspects of network analysis. One application of SNA is verification of MAS communication in a network of agents. For example, some works evaluate and verify the communication patterns of MAS. They classify communications in classes like overloader, overloaded, isolated, and regular. Then they try to find design drawbacks in terms of communication patterns [60–62]. Data mining tools are developed for this purpose. SNA techniques like clustering is used for code debugging [63]. Some other data analysis tools are available that analyze the interaction among agents and visualize the result of running simulations to be compared with the modeling of agents in MAS [64].

There is much difference between these works with what is presented in this paper. Many of the researches in this field are restricted to the verification of MAS from system logs, where the multiagent system is implemented and deployed. Some like [64] work in the design phase, but they use some kind of simulation and run the models of the system to analyze the agents' interactions. The other factor that makes a difference between our work and other researches that use SNA is analyzing the communication to verify or detect different violation factors. We mostly look for the violations in terms of new behaviors in the system (whether or not they are accepted/declined by designers and stakeholder). However, these researches do not detect possible new behaviors (referred to as emergent behavior in this article) and mostly look for communication performance, interaction verification, or issues associated with quality of service.

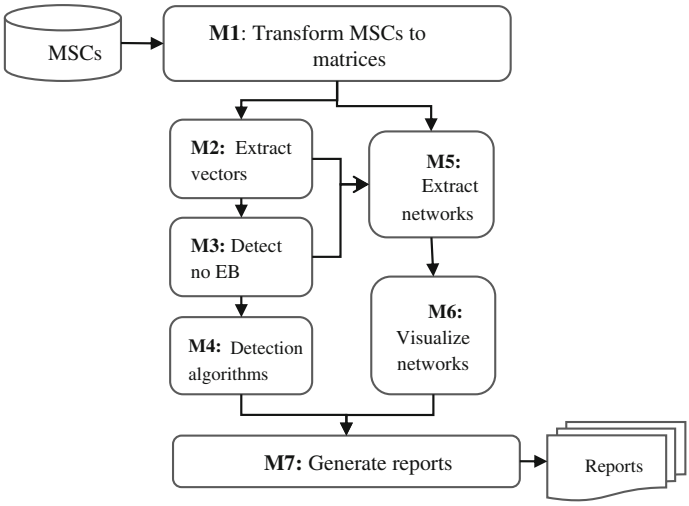


**Fig. 3** Main modules of the system

# 3 Methodology

## *3.1 System Architecture*

The main modules of the system are shown in Fig. 3. The MSCs are used to extract the allowable interactions among agents. The MSCs are transformed to interaction matrices in module M1. In the next step some vectors are extracted from the matrices in module M2. The required definitions can be found in next sections. In module M3 the agents with no emergent behavior are detected and removed from further analyses. The detection of agents with no emergent behavior helps in reducing the complexity of emergent behavior detection in system level. The algorithm works basically on the similarities of an agent's behavior in various MSCs to detect whether the agent is a safe agent (in terms of possibility to show an emergent behavior in component level). The behavior similarity of the agent is searched by finding all the interactions that one agent is involved in, the similarity of its states, and the messages the agent is involved in. The details of the algorithm and implementation are discussed in [21]. The next step is applying the emergent behavior algorithm in M4 for component and system level. The agents with no emergent behavior are omitted for further analysis in component level. Different algorithms are used in this module. Each algorithm detects an specific type of emergent behavior. On the right part, modules M5 and M6 are shown which extract networks and visualize data. Based on the algorithms in M4 and the networks extracted, M7 generates reports on the states, agents, and MSCs that an emergent behavior or implied scenario occurs. The report also contains information on the causes of this behavior which is also shown in the network.

The focus of this paper is mostly on M5 and M6 and the details of these two modules are presented. Therefore, details on the rest of the system are referred to other papers [19, 20].

### 3.2 Methodology Steps

The methodology for the detection of unexpected behavior has the following steps in general:

- Transfer the MSCs to interaction matrices.
- Transform message contents to message labels.
- Link message labels to entries of matrices.
- Extract vectors.
- Detect the agents that have no emergent behavior and omit them from processes in component level.
- Find shared states for each agent.
- Apply algorithms based on the type of emergent behavior.
- Prepare data.
- Visualize the networks.
- Apply network analysis and make them visible in the networks.
- Generate reports and compare results for the designer.

Some definitions of the matrices and the vectors, information about the networks, and how to prepare and visualize data are discussed in detail in the following for both classes of emergent behavior: component level and system level.

### 3.3 Component Level Analysis

In component level analysis, the detection of emergent behavior for one agent is important. In this level, an agent is considered and its states in all scenarios are investigated to detect the potential points in which the agent can be confused between two or more situations, or the states in which there is a branch with no priority to continue the next actions. This situation can be interpreted as a graph, which has an initial point, and the next node is followed when the required action for the associated edge happens in the system (events on the agent's life line). If one node has two or more different next nodes, either with same or different events on the associated edges, there is a branch in that graph. On one side, each branch should be taken into account with only one scenario of the system with required interaction among other agents. On the other side, when the software agent is traversing that graph, it has an option between the branches to choose and follow. Consequently, an unexpected situation can happen when the agent chooses another branch while the whole system expects the actions on a different branch to be performed by the agent, as it was

shown in the scenario of the system. In component level analysis, such branching choices are investigated to detect the emergent behaviors.

One possible reason of these branches in a graph is when some states of one agent are shared between two or more scenarios of the system. These states can be extracted from the interaction matrices (refer to Definition 1 for matrices). Since the state transitions are due to the messages sent/received by/to an agent, these states can form a network. The agent's states are the nodes of this network and the messages that cause a transition to another state are considered as the edges of the network. Visualizing the network of each agent's states can lead to the detection of emergent behaviors. This network, which is visualized as a directed graph, can be extracted using the interaction matrices. Each matrix is associated to one scenario, and therefore it contains the required information for a network state for each agent. Adding another scenario to the system means expanding this base network. As mentioned before, some states and their transitions may be shared among some scenarios. When expanding the network, these shared states can be visualized as the weights of a part of our network, i.e. when some states and the message labels are repeated weight of the edge between those states is increased. The more the weight, the thicker the edge is in the network. This visualization is helpful in conducting the research for detection of emergent behaviors in component level in a few aspects:

1. The shared states of each agent are shown in its states' network.
2. The branches in the graph of each agent are indicated.
3. The distance of these branches from the last state of the shared states are specified.
4. The message labels for the branching choices are illustrated.

All these choices are important when the agent behavior is verified. In case of having a branch in the network, two probable paths are:

1. Case C1: Going to two or more different states with the same message labels (edges)
2. Case C2: Going to two or more different states with different message labels (edges)

The former can be a design flaw if no condition is considered with the same messages. The latter is the case declared previously and can cause emerging a new behavior in the system. These two concerns can occur after the last state of the shared states or in case where there are no shared states between some scenarios.

It is worth mentioning that the agent should be a sender in all cases. It means the messages should be of type "sending" when the agent reaches a branching choice. These cases are demonstrated in the Fig. 4.

As it is shown in the figure, the branching occurs after state $S_1$. By receiving message label $m_1$ there are two available states that the agent can follow, $S_2$ and $S_3$. This is the path indicated in Case C1. The path in Case C2 happens when the next states can be obtained by receiving message labels $m_2$ and $m_3$ which goes to unique states $S_4$ and $S_5$ by following each of them.

Following our example of greenhouse MAS from introduction (Fig. 1), from $A_w$ point of view, and based on its states' graph, this agent has an option (branching

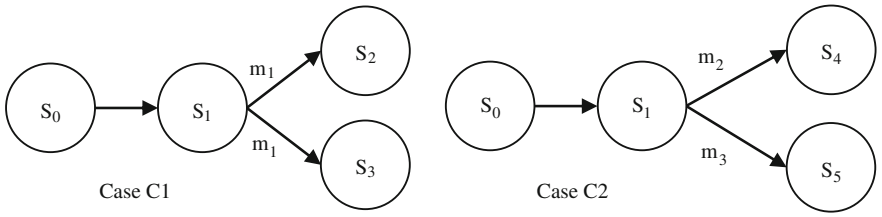


**Fig. 4** Branching choices

choice) to follow when it reaches its third state by receiving the message "Analyze". Therefore, the emergent behavior of Fig. 2 can happen.

## *3.4 System Level Analysis*

**Visualizing the Agent's Sequence of MSCs**.
System level analysis is referred to as detection of implied scenarios, when all components/agents of the system are performing their task and a new scenario is implied to the system. The system behavior should be considered in system level which means following the sequence of all scenarios in the hMSC while there is no central controller for the agents' actions in system level. This problem arises because the agents have local view from the system. In other words, each agent is independent of other agents when completing its actions and the agent has no control on its received messages. Consequently, they are not aware of the sequence of scenarios that other agents are performing. The result can be following different sequences of MSCs by various agents which results to have an implied scenario. Based on Uchitel, Kramer, and Magee's interpretation "*Implied scenarios are the result of specifying the global behaviors of system that will be implemented component wise*" [22]. This explanation is illustrated by an example from Uchitel and et al. paper [22]. The example in Fig. 5 consists of four components interacting in four MSCs. The overall behavior of the system is shown in an hMSC in right side of this figure.

An implied scenario that can happen in this example is shown in Fig. 6 and is taken from [22].

In this scenario, analysis is done based on the previous data that sensor has registered to database. However, the sensor is initiated, terminated, and then initiated again. But database, actuator, and control components analyze the first round of data from sensor component, before it was terminated. This is not acceptable based on MSCs and hMSCs.

For better investigation of this case, let's divide agents into two groups: *Passive* and *Active* agents. *Passive* agents are those agents who start an MSC by receiving a message from other agents. For example components DB and Actuator in our example

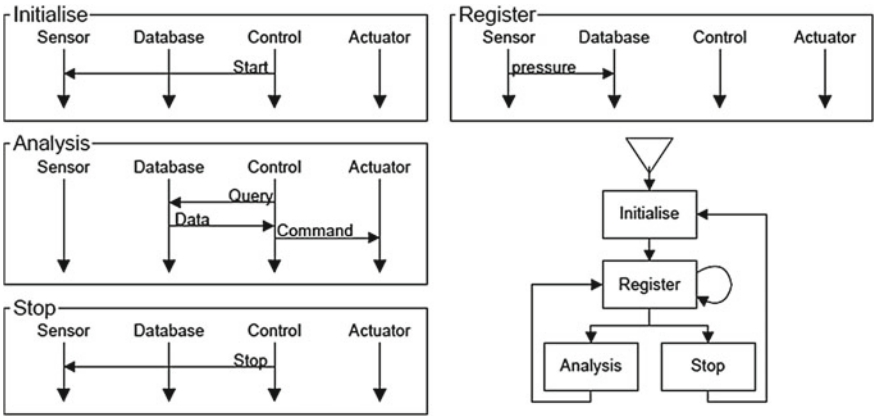


**Fig. 5** Four scenarios in the form of MSCs and the associated hMSC (Taken from Uchitel [22])

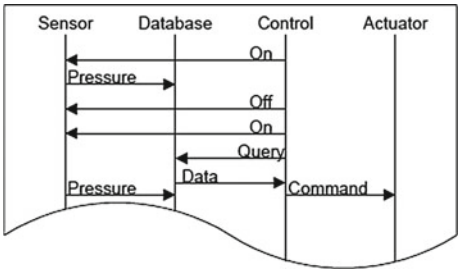


**Fig. 6** An implied scenario for system of Fig. 5 (Taken from Uchitel [22])

are *passive* components. *Active* agents like Control component in the example are defined as the agents who start an MSC by sending a message to other agents.

The sequence of MSCs followed by each component in the example is shown in Fig. 7. The name of each component is written above each graph. The abbreviations I, R, A, and T are used for MSC names Initialize, Register, Analysis, and Terminate (Stop) respectively. The dotted arrows show that the component has not involved in the associated MSCs. In other words, this component has not sent or received any messages in these MSCs, and therefore does not follow the paths between these MSCs in the hMSC. As an example, as it is obvious from Fig. 5, the Actuator has no communication with other components in the MSCs other than Analysis scenario. Therefore, there is no path between the MSCs in the Actuator's sequence of MSCs in Fig. 7. These sequences are derived from the MSCs and the hMSC, which can be extracted from the interaction matrices.

Based on what was derived as the sequence of MSCs for each component, the real sequence of MSCs for each component is shown in Fig. 8, where the dotted arrows are omitted. The name of the component is written on the left side of each graph. These graphs show that component Sensor has no involvement in Analysis scenario, DB component does not contribute in Initialize and Terminate MSCs, component

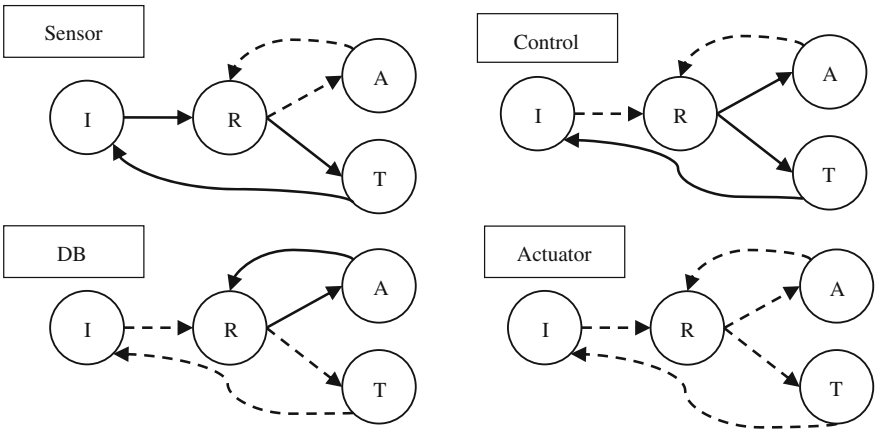


**Fig. 7** Sequence of MSCs for components of Fig. 5

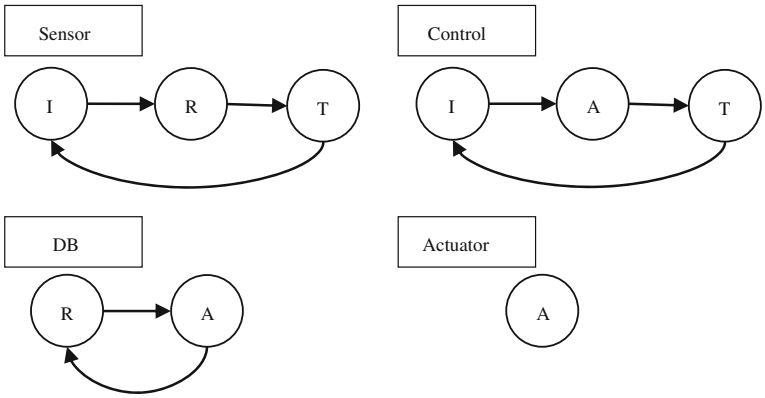


**Fig. 8** Sequence of MSCs demonstrating involving MSCs for each of components of Fig. 5

Control does not take part in Register MSC, and Actuator just has communications in Analysis MSC.

The graph of the agent's sequence of MSCs can be interpreted as an abstraction of the MSCs of the system from the agent's point of view. In other words, it shows how the agent realizes the hMSC to perform its actions. From the designer's point of view, the agent's sequence of MSCs can be interpreted as an abstraction of the agent's states in the whole system and in hMSC which shows how the MSCs are performed by that agent.

One possible path in the agent's sequence of MSCs that can lead to an implied scenario is the existence of loops in the internal MSCs of an hMSC. It can affect the system behavior, since the agent may perform the loop more than what is required by other agents, or when others agent are performing the next MSCs, this agent can continue the loop without considering a termination action and the whole system

behavior. DB component is an example of a component which has such a path in its sequence of MSCs.

The other possible path in an Active agent's sequence of MSCs is the existence of a path between internal MSCs of hMSC where the path does not include the initial and termination nodes of hMSC. In this case an implied scenario can exist since the agent is a sender in the startup of the MSCs it is involved in. Therefore, the agent considers the internal MSCs as the initial MSCs and can start at any point in the middle of hMSC without considering the position (hMSC nodes or MSCs) of other agents in the hMSC (i.e. without considering the MSCs that other agents are executing). This can be considered as having a sequence of MSCs like the ones related for DB or Actuator components, however, the components in our example are not Active.

**Visualizing the Network of Agents' Interactions**.
The other factor other than emergent behavior analysis and implied scenarios that can be investigated by visualization techniques is related to the communication performance. The visualization of the network of agents' interactions can help focusing on analysis of agents with specific characteristics and can lead to a clue for better design of the system. Regarding communication balancing and agents' performance, the agents' interaction network is used to analyze the betweenness and centrality for the agents. Agents with higher centrality/betweenness should be:

1. Analyzed for meeting the performance requirements.
2. Analyzed for emergent behavior analysis. Since the communication between these agents and other agents is high and they play a critical role in the system, any problem that arises with these agents can cause the whole system to fail. On the other hand, the problems in the behavior of these agents can propagate in the system and affect the system behavior.

These two points are especially important for the agents that play a coordinator role in inter-communication between various agent types in MAS.

## *3.5 Data Preparation*

As mentioned earlier, the MSCs and hMSC are the base inputs of the system. These two charts are used to indicate system agents and components involved, the interaction of these components to their environment and among themselves, general behavior of the agents and the whole system, and to what order the scenarios should act (i.e. system behavior).

For preparing our data for further analysis, these MSCs are translated to analyzable data structure, namely interaction matrices. In this technique, the scenarios in the form of MSCs are transformed to send/receive matrices. In synchronous communications (i.e. Messages are considered to be received at the same time they are sent), either send or receive matrices are applicable, because they are transposes of each other. Nevertheless, in asynchronous communications both send and receive matrices

should be utilized. The rest of this paper is based on considering the synchronous communication style. Definitions are followed:

**Matrix**. One $n \times n$ matrix for each scenario (MSC) is built where $n$ is the number of agents in the system.

**Definition 1** (*Send matrix*) Matrix $S$ is defined as the send matrix. $\mathrm{MSC_k}$ is the $k$th MSC of the system. For each $\mathrm{MSC_k}$ if there is a message sent from agent $A_i$ to agent $A_j$ the order of message as appeared in the scenario is entered as $s_{ij}$ entry of matrix $S_k$, otherwise $s_{ij} = 0$. If there is more than one message sent from one agent to another, the numbers are separated with a comma in that entry. The $S_k$ is the send matrix related to $\mathrm{MSC_k}$.

In this context each matrix is informative, since it shows the agents, their interactions, the interaction types (i.e. sending or receiving), and the order of messages. The only information missing is the message contents (not message labels).

For component level analysis, the states of each agent should be extracted from the Send matrix of Definition 1. The extracted states should contain information about the transitions between them, the message labels in this transition, and the type of message (Send or Receive). In the finite automata each message can transmit the agent from one state into another. This transition can be extracted from the matrices defined earlier. The state and state transition formal definitions are explained by Alur [6].

Considering the synchronous communication and send matrices, other definitions based on this basic definition can be outlined to help investigate the emergent behavior detection process and show the specific information about agents.

The entities of S matrix in row and column $i$ are the related message numbers for $A_i$. For each agent $A_i$ in matrix $S_k$ the entities $s_{is}$ show the entities in row $i$ and column $s$ where $1 \leq s \leq n$ and $n$ is the total number of agents in the system. This means the message numbers that agent $A_i$ has sent to other agents and/or to itself. Also the entities $s_{ri}$ show the entities in column $i$ and rows $r$ where $1 \leq r \leq n$. This means the message numbers that agent $A_i$ has received in the related $\mathrm{MSC_k}$. The order of the numbers in the row and column $i$ shows the order of states for that agent. In the case that the agent had sent a message to itself, two states are considered, one for sending and the other for receiving a message. The row number other than $i$ show the agents that send messages to agent $A_i$.

**Definition 2** (*Agent send communication vector* ) Vector $v_{isk} = (\mathrm{e_{i1}}, \mathrm{e_{i2}}, \ldots, \mathrm{e_{in}})$ shows the vector of row $i$ in matrix $S_k$ and $n$ is the total number of agents. When there is more than one number in one entry of the S matrix, the element is considered as a compound element. It means for example if there is entry $\mathrm{s_{i1}} = 2, 3$ in matrix $S_k$; then in the vector $v_{isk}$ element $\mathrm{e_{i1}}$ will be $\mathrm{e_{i1}} = (2, 3)$. This vector shows all of the communications of agent $A_i$ to other agents in the $k$th MSC where $A_i$ is the sender of the messages.

**Definition 3** (*Agent receive communication vector*) Vector $v_{rik} = (\mathrm{e_{1i}}, \mathrm{e_{2i}}, \ldots, \mathrm{e_{ni}})$ shows the vector of column $i$ in matrix $S_k$ and $n$ is the total number of agents. When there is more than one number in one entry of the S matrix, the element is considered

as a compound element. This vector shows all of the communications of agent $A_i$ to other agents in the $k$th MSC where $A_i$ is the receiver of the messages.

The state vector of each agent $A_i$ in $MSC_k$ is defined as:

**Definition 4** (*Agent state vector*) The states of each agent $A_i$ in $k$th MSC is shown by a vector $states_{ik}$ which is the ordered vector of the entities of $v_{isk}$ and $v_{rik}$ where the zero entities are omitted and the elements are in ascending order. In the case that an agent had sent a message to itself, the same numbers in the $v_{isk}$ and $v_{rik}$ are added distinctly to the states vector. Therefore two states are considered for this agent by sending a message to itself: one for sending and one for receiving the message. The number of the elements in vector $states_{ik}$ shows the total number of events on agent $A_i$ in MSC $k$ or the projection of events of $k$th MSC on $A_i$ ($\mathrm{MSC_k}|_{\mathrm{Ai}}$). Elements of vector $states_{ik} = (st_1, st_2, \ldots, st_{f-1}, st_f)$ are the states of agent $A_i$. The transitions between the states are by messages sent or received to/by agent $A_i$. The last element $st_f$ shows the accepted state of agent $A_i$ in MSC $k$.

Note that since each element of the state vector $states_{ik}$ matches an element in $v_{isk}$ or $v_{rik}$, we can trace the source entity $s_{kp}$ in the related $S$ matrix that it matches. Therefore, the type of the message or state can be recognized based on whether it matches a send event or a receive one.

The advantage of using these matrices is that the information about the agent that causes the state transition for each individual agent can be identified by tracking the number of that element in each of the $v_{isk}$ or $v_{rik}$ vectors.

**Definition 5** (*State transition sender vector*) The state transition sender vector for agent $A_i$ in MSC $k$ is shown by $Sender_{ik} = (Snd_1, Snd_2, \ldots, Snd_p)$ where each element shows the sender for a message that changes the state for agent $A_i$. Each element of $states_{ik}$ has a matching either in $v_{isk} = (\mathrm{e_{i1}}, \mathrm{e_{i2}}, \ldots, \mathrm{e_{in}})$ or in $v_{rik} = (\mathrm{r_{1i}}, \mathrm{r_{2i}}, \ldots, \mathrm{r_{ni}})$. Therefore, the senders of the messages for each state are either agent $A_i$ (if the element in $states_{ik}$ has a matching in $v_{isk}$) or are the agent $A_w$ (if the element in $states_{ik}$ has a matching with the $w$th element in $v_{rik}$). Simply explaining, this vector shows which agent is the sender of each event for agent $A_i$ in MSC $k$.

**Greenhouse System Example**.
The scenarios in Fig. 1 are a part of scenarios of a greenhouse multiagent system. This system consists of three different types of agents: Temperature balancing (At), Water control (Aw), and Mineral control (Am) agents. 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. The system is designed using MaSE methodology. MSCs are extracted from the sequence diagrams of MaSE and the agent classes (MaSE artifacts) as explained in [26]. One method for temperature balancing is shown in MSC1 in Fig. 9. The $\mathrm{S_t}$ and $\mathrm{S_m}$ show the temperature and mineral sensors respectively and KB stands for knowledge base. MSC2 in Fig. 10 shows the request of $\mathrm{A_m}$ to the water control agent to serve minerals for the plants. For simplicity, just two of the MSCs of the system are shown here.

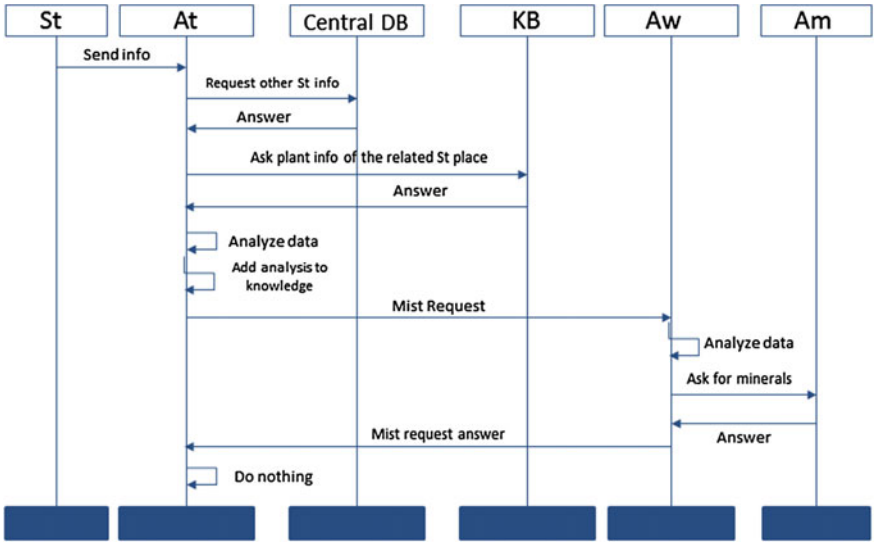


**Fig. 9** MSC1: Temperature balancing method

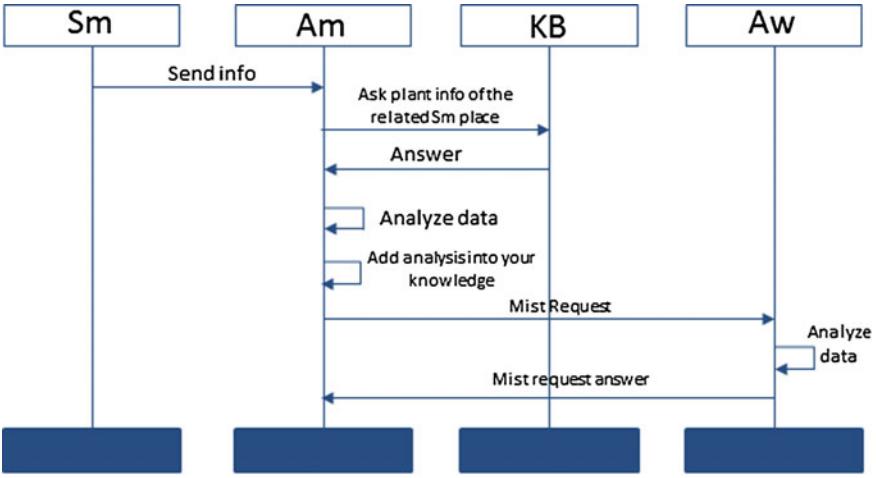


**Fig. 10** MSC2 Mineral balancing method

*Extract Matrices and Vectors for Emergent Behavior Detection*

The first step is to transform the MSCs to related $S$ matrices which show the send matrices of MSCs. As an example, the matrix $S_1$ related to the MSC1 is shown below:

$$\begin{bmatrix} & S_t & A_t & DB & KB & A_w & A_m & S_m & th \\ S_t & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ A_t & 0 & 6.7 & 2 & 4 & 8 & 0 & 0 & 13 \\ DB & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ KB & 0 & 5 & 0 & 0 & 0 & 0 & 0 & 0 \\ A_w & 0 & 12 & 0 & 0 & 9 & 10 & 0 & 0 \\ A_m & 0 & 0 & 0 & 0 & 11 & 0 & 0 & 0 \\ S_m & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ th & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

In the MSCs, we will consider the rows and columns ($i = 5$) related to $A_w$. For this agent in the MSCs, the vectors $v_{5sk}$ and $v_{r5k}$ show the related row and column of $A_w$ and $states_{5k}$ represents the associated states vector of $A_w$ in the related MSCs, where $k$ shows the $k$th MSC (MSC$_k$). The states vectors of $A_w$ are:

$States_{51} = \{st_1, st_2, st_3, st_4, st_5, st_f\}$
$States_{52} = \{st_1, st_2, st_3, st_f\}$

And the associated message labels to each of the state vectors are:

$mLabels_{51} = \{\mathrm{m}_1, \mathrm{m}_2, \mathrm{m}_2, \mathrm{m}_3, \mathrm{m}_4, \mathrm{m}_5\}$
$mLabels_{52} = \{\mathrm{m}_1, \mathrm{m}_2, \mathrm{m}_2, \mathrm{m}_6\}$

For this agent, the state transition sender vectors are shown with $Sender_{5k}$ in the $k$th MSC. The sender vectors are:

$Sender_{51} = \{\mathrm{A}_t, \mathrm{A}_w, \mathrm{A}_w, \mathrm{A}_w, \mathrm{A}_m, \mathrm{A}_w\}$
$Sender_{52} = \{\mathrm{A}_m, \mathrm{A}_w, \mathrm{A}_w, \mathrm{A}_w\}$

The state's network of $A_w$ for these two MSCs is shown in Fig. 11 with message labels and senders of each state transition for each of them is shown above them. The initial and final states are differentiated.

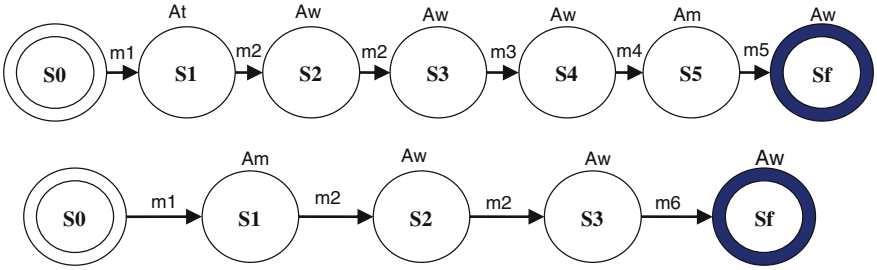


**Fig. 11** Agent Aw's states network for MSC1 and MSC2

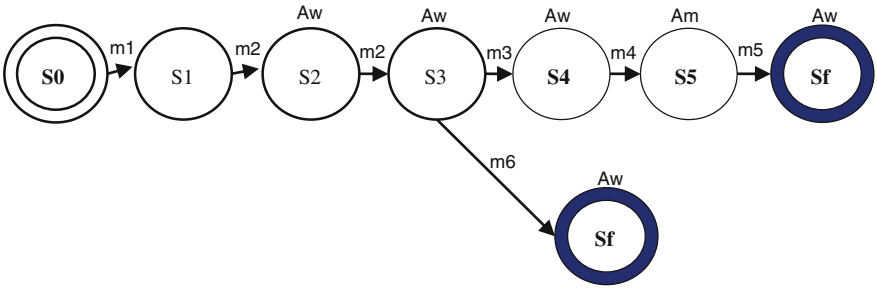


**Fig. 12** Visualized states' network of Aw for two MSCs

## *3.6 Data Visualization*

Base of the agent's states network is constructed by one of its state vectors. Other nodes are added to this graph from other agent's state vectors either by adding a separate node when the agent's state is different or by merging to the existing graph when some states are the same with existing nodes. The weights of the edges for the shared parts are increased based on the times of repetition. This can be visualized by bolding the edges or by color changes for edges' weights. Note that shared states can create various branches to the graph. The state transitions or messages are the edge labels for this network.

Continue our example of greenhouse system, the states' network of $A_w$ can form like the one in Fig. 12. Since a part of the states' vectors in the two MSCs are the same, they share part of the graph and then create a branch in the states' network. The shared states and edges are bolded. However, this is not a proof for being same states regarding the states, messages labels, and sender transitions. As it is shown in the figure, state $s_1$ shows no sender name, because the senders in the two MSCs are different for this state.

For system level and visualization of agents' interaction network, the matrices in Definition 1 are used. The entries of matrices show all of the communications between the agents in one MSC. The number of the messages sent to each agent can be extracted from the same matrix and depends on the associated entry. Other matrices are used later to specify more interactions or change the network based on the number of messages sent to other agents. The weight of edges is defined as the number of messages communicated between two agents.

## *3.7 Verification*

**Verifying New Paths Against MSCs.**
At component level there are branching choices that visualized in previous sections based on the agents' states. These branches may cause an emergent behavior in the system. However, not all of the branching choices lead the agent to emerge a new behavior. The visualization technique acts as an indication to the possible points of

such behaviors for an agent. Therefore, they should be verified against the MSCs and defined system behavior to detect violations. When we traverse the graph from the initial node and follow the other nodes through connecting edges, by following each branch we will have various paths to reach the final state (last node of the directed graph). These paths can be identified by nodes and edges, states and messaged labels respectively, associated to the state transition senders. These senders can be extracted by the vectors in Definition 5. Therefore, all paths include the senders' information as well.

Verification of each branch is by determining whether or not the path that contains the branch is among the agent's state vectors with respect to both *message labels* and *state transition senders*.

Consider the emergent behavior of greenhouse system shown in Fig. 2. The allowed paths for $A_w$ in MSC1 and MSC2 are:

$$\begin{aligned} Acceptable\_Path_{51} &= \{st_1, m_1, A_t; st_2, m_2, A_w; st_3, m_2, A_w; st_4, m_3, A_w; st_5, m_4, \\ &\qquad A_m; st_6, m_5, A_w\} \\ Acceptable\_Path_{52} &= \{st_1, m_1, A_m; st_2, m_2, A_w; st_3, m_2, A_w; st_4, m_6, A_w\} \end{aligned}$$

The elements are separated by a semicolon, and each element consists of a tuple (*st, m, A*) where *st* shows the state of the agent, *m* stands for associated message label, and the sender transition for that state is shown by *A*. However, the paths that are extracted from the network contain other paths:

$$\begin{aligned} New\_Path_1 &= \{st_1, m_1, A_t; st_2, m_2, A_w; st_3, m_2, A_w; st_4, m_6, A_w\} \\ New\_Path_2 &= \{st_1, m_1, A_m; st_2, m_2, A_w; st_3, m_2, A_w; st_4, m_3, A_w\} \end{aligned}$$

As it was mentioned before, this is a new path which is not verified based on the allowable paths extracted from MSCs. Therefore, an emergent behavior can happen in the system.

**Verifying MSC Sequences Against hMSC**.
Two possible conditions that can lead to an implied scenario were investigated previously. The first one is the existence of loops in the internal MSCs of an hMSC in the agent's sequence of MSCs. The other condition is the existence of a path between internal MSCs of hMSC in the agent's sequence of MSCs where the path does not include the initial and termination nodes of hMSC for an *Active* agent. These two conditions can simply be detected and verified.

However, there is another condition that leads to an implied scenario and needs to be verified against the hMSC. These can be verified by finding the paths in agent's sequence of MSCs and verifying it against the hMSC. For example, consider the hMSC of Fig. 4. In this example, the MSC sequences extracted from the hMSC and MSCs for control component shows a path of MSCs as follows: Initialize, Analysis, and Terminate, and then from Terminate MSC it can go back to Initialize MSC again. This is shown in Fig. 7. However, when comparing this sequence of MSC {Initialize, Analysis, Terminate} (Fig. 8) there is not such a sequence in hMSC, because there is no direct path from Analysis MSC to Terminate MSC. When verifying the control sequence of MSCs against the hMSC, this contradiction is detected which can lead to another implied scenario in the system.

### *3.8 Generating Report*

The report is generated to inform the designer about the possible emergent behaviors and implied scenarios, the involved agents, the reasons of occurring such behaviors, and the MSCs and states that have conflicts with the MSCs and hMSC of the system. This report helps to announce the problematic points and leads to solve the problem as well.

As an example, consider the greenhouse system. The emergent behavior explained previously is because of the conflicts of the new path with the accepted paths in the MSCs. The new paths contain the same states, message labels, and sender transitions for most of its elements. However, two conflicts are found. The first conflict is in the first element of two new paths $(st_1, \mathrm{m}_1, \mathrm{A}_\mathrm{t})$ or $(st_1, \mathrm{m}_1, \mathrm{A}_\mathrm{m})$ which has a different sender, comparing to similar accepted paths from the system specifications (MSCs). The other conflict is in the message label of the last element of these paths $(st_4, \mathrm{m}_6, \mathrm{A}_\mathrm{w})$ or $(st_4, \mathrm{m}_3, \mathrm{A}_\mathrm{w})$. These two conflicts show that either the senders of first states should be the same for two MSCs or the message labels of the last element should be changed, based on the message labels of the acceptable paths and the shared states of all paths. The report contains all this information which suggests a solution as well. When the changes are defined the new MSC specifications should be checked again to have consistency.

## 4 Discussion

Social network analysis has various applications. One of the applications of them is shown in this paper through modeling the agents' interaction into various interaction network and analysis of these networks for system verification. The network visualization explained in this paper is used for MAS verification against emergent behavior and implied scenarios. The aim of this work is the detection of these behaviors and whether or not they are accepted or declined depends on the designers and stakeholders. However, these emergent behaviors should be detected earlier in the system to prevent or reduce later damage and costs.

The agent's states' network may seem similar to the state machines. However, they are semantically different from state machines. State machines are not always shown to the designer and the process of behavioral modeling is also different. The alphabet, words, and languages are important in state machines; while in agent's states' network other features are used. Some information about the senders, message types, and transitions of each state are preserved in the network which makes the specification, features, and applications of these networks different from state machines.

One major discussion about emergent behavior detection methodologies is whether they can detect all the unexpected behaviors or not. For our methodology, we are currently working on classification of types of emergent behaviors in these systems. By classification of various scenarios that can happen in a system, we can investigate

this issue, and also suggest solutions to fix these behaviors. The solutions that are suggested and the visualization of the problematic parts are among contributions of this work that are not present in the existing works. As a basic verification of our methodology, until now, all the implied scenarios of other works for instance for the example of Uchitel's case [22] are detected by our methodology. Furthermore, some other issues that can result in an implied scenario are presented which were not discussed in other works. However, in this paper the correctness or completeness of our methodologies is not mentioned.

One of the advantages of our methodology is transforming MSCs to interaction matrices without underlying semantics, which is inspired by social network analysis techniques. This abstraction from the semantics is valuable in terms of removing synthesis phase for emergent behavior detection which is mentioned to be more complex than the model checking. Model checking has problems like scalability for large scale systems. Some proofs for providing scalability by this methodology is applying techniques to detect agents with no emergent behavior in early steps and omit them from further analysis, which helps improve the scalability of the system. The other issue is that matrices have $n^2$ objects where most of them are zero entities. This is solved by the definitions when extracting appropriate vectors for analysis. The vectors contain just non-zero elements which is much less than $n^2$. Therefore the method has neither the analysis of $n^2$ objects nor the zero entities for scalability problem. It is worth mentioning that the analysis for detection of agents with no emergent behavior became feasible by defining interaction matrices rather than using formal method and state machine approaches. This is another advantage of using network analysis in the verification of DSS and MAS.

The other contribution is on visualizing the results which is the main focus of this paper. Based on our knowledge, the three networks discussed in this paper and the visualization techniques for this problem are not indicated in other works. The visualization of these analyses for the network of agents and their states helps the designer for faster fixing of the represented results for detected emergent behaviors. It helps not only see what and where in the system, but also how the emergent behavior is happening. This is another contribution of the work. Most of the existing methodologies detect the emergent behavior without leading the reasons or a solution for it [17]. The generated reports contain all the information about the detected emergent behavior or implied scenario, and therefore covering the shortage of the existing approaches.

The major difference of our work with other researches that try to find emergent behaviors and implied scenarios is using network mining rather than modeling with formal methods. In our work, we try to model the interaction of system components in networks. Then we define the reasons of emerging new behaviors and find criteria and restrictions for the extracted networks. These criteria are used as the main verification method for the extracted networks to investigate and verify them against having emergent behaviors and implied scenarios. Since the modeling is quite different in our work and other researches, the comparison criteria can be quite different. Table 1 represents a comparison between existing main approaches in this field and our work. Some criteria in this table are the level of analysis and whether or not they are capable

**Table 1** Comparison of other approaches and our work

| Research/ criteria | Source diagram | Modeling | CLA[a] | SLA[b] | Origins of problem[c] | Visualize results | Suggest solution |
|---|---|---|---|---|---|---|---|
| Uchitel et al. [8, 22, 32] | MSC/ hMSC | Labeled Transition System (LTS) | No | Yes | No | LTS | X |
| Genest et al. [25][d] | MSC/ hMSC/MSG | State machines | Yes | Yes | X | X | X |
| Whittle [30] | UML SD | Statechart | Yes | Yes | X | X | X |
| Mousavi [4] | MSC/ hMSC | Finite state machine | Yes | Yes | X | X | X |
| Alur et al. [6][e] | MSC/ hMSC/MSG | Automata temporal logic formulas | Yes | Yes | X | X | X |
| Song et al. [47] | UML SD | Causal graphs | X | Yes | Yes | Orders/ Causal graphs | X[f] |
| Our work | MSC/ hMSC | Interaction networks/ MSC sequences | Yes | Yes | Yes | Yes | Yes |

[a]CLA is used as an abbreviation for Component Level Analysis
[b]SLA is used as an abbreviation for System Level Analysis
[c]This criteria means that the source of the problem is detected and shown to the designer rather than just the existence of an emergent behavior is notified. This is due to the exact cause of emergent behavior both in system and component level analysis, as mentioned in literature, is the local view or restricted view of the components of the system
[d]This paper and their recent papers of the authors mostly discuss the MSC/hMSC specification languages and the validation problems about model-checking and implementability
[e]This paper and related papers of the authors mostly discuss on the computations and validation features of MSC/hMSC/MSG. They discuss the decidability and implementability as well as complexity of model checking
[f]They just provide the parts of the system that should be considered to handle the implied scenarios

of visualizing the analysis results in an effective way to the designer (i.e. in a way that leads to a solution or revising the designs).

## 5 Conclusion and Future Work

Taking advantage of visualization techniques is an important factor in verification of MAS for detection of emergent behaviors and implied scenarios. The networks defined and presented in previous sections prove the role of this technique in the verification field. These networks are used both for component level and system

level analysis and demonstrates the wide application of network visualization in MAS verification. Although the definitions are verified on MAS, the methodology and visualization technique can be used for verification of DSS as well. Since in this approach we mostly deal with the behavior of the system in terms of MSCs and hMSC and not the internal knowledge or behavior of agents. Consequently, the terms used in this article can be used for DSS verification as well.

Two main future lines of works are in our plans. First, demonstrating various elements such as message types, type of states, and whether the agent is a sender after branching choice in its states' network that can be added as visualization options to the system. Second, bolding the shared states and color differentiation among the shared states with various sender transitions can be illustrated in the network. These features make the designer aware of possible problems in advance before analyzing and verifying the whole system. Other possible future directions include considering other measurements for nodes and links and investigating the effectiveness of those features in the software verification problem.

**Acknowledgments** 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

1. Wooldridge M, Fisher M, Huget M-H, Parsons S (2002) Model checking multi-agent systems with MABLE. In: Proceedings of the first international joint conference on autonomous agents and multiagent systems: part 2. ACM: Bologna, pp 952–959
2. Bordini R, Fisher M, Pardavila C, Visser W, Wooldridge M (2003) Model checking multi-agent programs with CASP. In: Hunt W Jr, Somenzi F (eds) Computer aided verification. Springer, Berlin, pp 110–113
3. Moshirpour M, Mousavi A, Far BH (2010) A technique and a tool to detect emergent behavior of distributed systems using scenario-based specifications. In: 22nd IEEE international conference on tools with artificial intelligence (ICTAI)
4. Mousavi A (2009) Inference of emergent behaviours of scenario-based specifications. In: Department of electrical and computer engineering. University of Calgary, Calgary, p 160
5. Broy M (2011) Seamless method- and model-based software and systems engineering. The future of software engineering. Springer, Berlin, pp 33–47
6. Alur R, Etessami K, Yannakakis M (2003) Inference of message sequence charts. IEEE Trans Softw Eng 29(7):623–633
7. Chakraborty J, D'Souza D, Narayan Kumar K (2010) Analysing message sequence graph specifications. In: Margaria T, Steffen B (eds) Leveraging applications of formal methods, verification, and validation. Springer, Berlin, pp 549–563
8. Uchitel S (2003) Incremental elaboration of scenario-based specifications and behaviour models using implied scenarios. In: Department of computing. Imperial College of Science, Technology and Medicine, University of London, p 173
9. Chaki S, Clarke E, Grumberg O, Ouaknine J, Sharygina N, Touili T, Veith H (2005) State/event software verification for branching-time specifications. In: Romijn J, Smith G, Pol J (eds) Integrated formal methods. Springer, Berlin, pp 53–69
10. Sanchez E, Squillero G, Tonda A (2012) Automatic software verification. Industrial applications of evolutionary algorithms. Springer, Berlin, pp 17–30

11. Ammar K, Pullum L, Taylor B (2006) Augmentation of current verification and validation practices. Methods and procedures for the verification and validation of artificial neural networks. Springer, New York, pp 13–31
12. Quan TT, Hoang DLN, Nguyen BT, Nguyen AN, Tran QD, Nguyen PH, Bui TH, Do AT, Huynh LV, Doan NT, Huynh NT, Nguyen TD, Nguyen TT, Nguyen VH et al (2010) MAFSE: a model-based framework for software verification, pp 150–156
13. Verhulst E, Jong G, Mezhuyev V (2008) An industrial case: pitfalls and benefits of applying formal methods to the development of a network-centric RTOS. In: Cuellar J, Maibaum T, Sere K (eds) FM 2008: formal methods. Springer, Berlin, pp 411–418
14. Briand LC (2010) Software verification—a scalable, model-driven, empirically grounded approach. Simula Research Laboratory. Springer, Berlin, pp 415–442
15. Knight J (1998) Challenges in the utilization of formal methods. In: Ravn A, Rischel H (eds) Formal techniques in real-time and fault-tolerant systems. Springer, Berlin, pp 1–17
16. Kneuper R (1997) Limits of formal methods. Form Asp Comput 9(4):379–394
17. Song I-G, Sang-Uk J, Ah-Rim H, Doo-Hwan B (2011) An approach to identifying causes of implied scenarios using unenforceable orders. Inf Softw Technol 53(6):666–681
18. Bontemps Y, Schobbens P-Y (2007) The computational complexity of scenario-based agent verification and design. J Appl Log 5(2):252–276
19. Fard FH, Far BH (2013) Detection and verification of a new type of emergent behavior in multiagent systems. In: 17th international conference on intelligent engineering systems (INES)
20. Fard FH (2013) Detecting and fixing emergent behaviors in distributed software systems using a message content independent method. In: Press in 28th IEEE/ACM international conference on automated software engineering (ASE), Doctoral symposium. IEEE
21. Fard FH, Far BH (2012) A method for detecting agents that will not cause emergent behavior in agent based systems—a case study in agent based auction systems. In: 2012 IEEE 13th international conference on information reuse and integration (IRI)
22. Uchitel S, Kramer J, Magee J (2002) Implied scenario detection in the presence of behaviour constraints. Electron Notes Theor Comput Sci 65(7):65–84
23. Carrol JM (1999) Five reasons for scenario-based design. In: Proceedings of the 32nd annual Hawaii international conference on system sciences, HICSS-32
24. Lunjin L, Dae-Kyoo K (2011) Required behavior of sequence diagrams: semantics and refinement. In: 2011 16th IEEE international conference on engineering of complex computer systems (ICECCS)
25. Genest B, Muscholl A (2005) Message sequence charts: a survey. In: Fifth international conference on application of concurrency to system design, ACSD 2005
26. Mani N, Garousi V, Far BH (2008) Monitoring multi-agent systems for deadlock detection based on UML models. In: Canadian conference on electrical and computer engineering, CCECE 2008
27. Genest B, Muscholl A, Peled D (2004) Message sequence charts. In: Desel J, Reisig W, Rozenberg G (eds) Lectures on concurrency and petri nets. Springer, Berlin, pp 103–121
28. Broy M (2000) The essence of message sequence charts. In: Proceedings of the international symposium on multimedia software engineering 2000
29. Krüger IH (2000) Distributed system design with message sequence charts. In: Institute of computer science 2000, Technical University of Munich, p 386
30. Whittle J, Schumann J (2006) Scenario-based engineering of multi-agent systems in agent technology from a formal perspective. In: Rouff C et al (eds) Springer, London, pp 159–189
31. UNION-, TSSOI-IT, SERIES Z (2004) Languages and general software aspects for telecommunication systems—formal description techniques (FDT)—message sequence chart, ITU-T Recommendation Z.120, p 136
32. Uchitel S, Kramer J, Magee J (2001) Detecting implied scenarios in message sequence chart specifications. In: Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on foundations of software engineering 2001. ACM, Vienna, pp 74–82

33. Alur R, Yannakakis M (1999) Model checking of message sequence charts. In: Baeten JM, Mauw S (eds) Concurrency theory CONCUR'99. Springer, Berlin, pp 114–129
34. Moshirpour M (2011) Model-based detection of emergent behavior in distributed and multi-agent systems from component level perspective. In: Department of electrical and computer engineering 2011, University of Calgary, Calgary, p 109
35. Muccini H (2003) Detecting implied scenarios analyzing non-local branching choices. In: Pezzè M (ed) Fundamental approaches to software engineering. Springer, Berlin, pp 372–386
36. Sousa FCd, Mendonca NC, Uchitel S, Kramer J (2007) Detecting implied scenarios from execution traces. In: Proceedings of the 14th working conference on reverse engineering 2007. IEEE Computer Society, pp 50–59
37. Uchitel S, Kramer J, Magee J (2003) Synthesis of behavioral models from scenarios. IEEE Trans Softw Eng 29(2):99–115
38. Adsul B, Mukund M, Kumar KN, Narayanan V (2005) Causal closure for MSC languages. In: Sarukkai S, Sen S (eds) FSTTCS 2005: foundations of software technology and theoretical computer science. Springer, Berlin, pp 335–347
39. Bhateja P, Gastin P, Mukund M, Kumar KN (2007) Local testing of message sequence charts is difficult, in fundamentals of computation theory. In: Csuhaj-Varjú E, Ésik Z (eds) Springer, Berlin, pp 76–87
40. Alur R, Etessami K, Yannakakis M (2005) Realizability and verification of MSC graphs. Theor Comput Sci 331(1):97–114
41. Alur R, Yannakakis M (2001) Model checking of hierarchical state machines. ACM Trans Program Lang Syst 23(3):273–303
42. Schumann J, Whittle J (2001) Automatic synthesis of agent designs in UML. In: Rash J et al (eds) Formal approaches to agent-based systems. Springer, Berlin, pp 148–162
43. Whittle J, Kwan R, Saboo J (2005) From scenarios to code: an air traffic control case study. Softw Syst Model 4(1):71–93
44. Whittle J, Schumann J (2000) Generating statechart designs from scenarios. In: Proceedings of the 22nd international conference on software engineering 2000. ACM, Limerick, pp 314–323
45. Ben-Abdallah H, Leue S (1998) MESA: Support for scenario-based design of concurrent systems. In: Steffen B (ed) Tools and algorithms for the construction and analysis of systems. Springer, Berlin, pp 118–135
46. Ben-Abdallah H, Leue S (1997) Syntactic detection of process divergence and non-local choice in message sequence charts. In: Brinksma E (ed) Tools and algorithms for the construction and analysis of systems. Springer, Berlin, pp 259–274
47. Song I-G, Jeon SU, Bae DH (2009) A graph based approach to detecting causes of implied scenarios under the asynchronous and synchronous communication styles. In: Proceedings of the 16th Asia-Pacific software engineering conference 2009. IEEE Computer Society, pp 53–60
48. Uchitel S, Kramer J (2001) A workbench for synthesising behaviour models from scenarios. In: Proceedings of the 23rd international conference on software engineering 2001. IEEE Computer Society, Toronto, Ontario, pp 188–197
49. Letier E, Kramer J, Magee J, Uchitel S (2008) Deriving event-based transition systems from goal-oriented requirements models. Autom Softw Eng 15(2):175–206
50. Letier E, Kramer J, Magee J, Uchitel S (2005) Monitoring and control in scenario-based requirements analysis. In: Proceedings of the 27th international conference on software engineering 2005. ACM, St. Louis, pp 382–391
51. Magee J, Pryce N, Giannakopoulou D, Kramer J (2000) Graphical animation of behavior models. In Proceedings of the 22nd international conference on software engineering 2000, ACM, Limerick, pp 499–508
52. Henriksen J, Mukund M, Kumar KN, Thiagarajan PS (2000) Regular collections of message sequence charts. In: Nielsen M, Rovan B (eds) Mathematical foundations of computer science 2000. Springer, Berlin, pp 405–414
53. Mukund M, Kumar KN, Sohoni M (2000) Synthesizing distributed finite-state systems from MSCs. In: Palamidessi C (ed) CONCUR 2000—concurrency theory. Springer, Berlin, pp 521–535

54. Aggarwal C (2011) An introduction to social network data analytics. In: Aggarwal CC (ed) Social network data analytics. Springer, New York, pp 1–15
55. Sabater J, Sierra C (2002) Reputation and social network analysis in multi-agent systems. In: Proceedings of the first international joint conference on autonomous agents and multiagent systems: part 1. ACM, Bologna, pp 475–482
56. Hanneman RA, Riddle M (2005) Introduction to social network methods. http://faculty.ucr.edu/~hanneman/nettext/C6_Working_with_data.html
57. Mislove AE (2009) Online social networks: measurement, analysis, and applications to distributed information systems. Rice University
58. Song HH, Cho TW, Dave V, Zhang Y, Qiuet L (2009) Scalable proximity estimation and link prediction in online social networks. In: Proceedings of the 9th ACM SIGCOMM conference on internet measurement conference 2009. ACM, Chicago, pp 322–335
59. Sun J, Tang J (2011) A survey of models and algorithms for social influence analysis. In: Aggarwal CC (ed) Social network data analytics. Springer, New York, pp 177–214
60. Gutiérrez C, García-Magariño I, Fuentes-Fernández R (2011) Detection of undesirable communication patterns in multi-agent systems. Eng Appl Artif Intell 24(1):103–116
61. Gutierrez C, García-Magariño I (2011) Revealing bullying patterns in multi-agent systems. J Syst Softw 84(9):1563–1575
62. Gutiérez C, García-Magariño I, Gómez-Sanz J (2009) Evaluation of multi-agent system communication in INGENIAS. In: Cabestany J et al (eds) Bio-Inspired systems: computational and ambient intelligence. Springer, Berlin, pp 619–626
63. Botía J, Hernansáez J, Gómez-Skarmeta A (2007) On the application of clustering techniques to support debugging large-scale multi-agent systems. In: Bordini R et al (eds) Programming multi-agent systems. Springer, Berlin, pp 217–227
64. Botía J, Gómez-Sanz J, Pavón J (2006) Intelligent data analysis for the verification of multi-agent systems interactions. In: Corchado E et al (eds) Intelligent data engineering and automated learning—IDEAL'06. Springer, Berlin, pp 1207–1214