Model Based Detection of Implied Scenarios in Multi-Agent Systems

Mohammad Moshirpour[1], Shimaa M. El-Sherif[1],
Abdolmajid Mousavi[2], Behrouz H. Far[1]

[1] *Department of Electrical and Computer Engineering,
University of Calgary, Calgary, Alberta, Canada
[1]{mmoshirp, smmelshe, far}@ucalgary.ca*

[2] *Electrical and Computer Group,
University of Lorestan, Khorram Abad, Iran
[2]mousavi@lu.ac.ir*

Abstract

Multi-agent systems (MAS) are efficient solutions for commercial applications such as information retrieval and search. In a MAS, agents are usually designed with distribution of functionality and control. Lack of central control implies that the quality of service of MAS may be degraded because of possible unwanted behavior at the runtime, commonly known as emergent behavior. Detecting and removing emergent behavior during the design phase of MAS will lead to huge savings in deployment costs of such systems. An effective approach for the MAS design is to describe system requirements using scenarios. A scenario, commonly known as a message sequence chart or a sequence diagram, is a temporal sequence of messages sent between agents. In this research a method for detecting emergent behavior of MAS by detecting incompleteness and partial description of scenarios is proposed. This method is explained along with a prototype MAS for semantic search that blends the search and ontological concept learning. Finally a devolved tool which automates the proposed methodologies is presented.

Keywords: *multi-agent system, emergent behaviour, message sequence chart, semantic search, annotation.*

1. Introduction

There are several agent oriented software engineering (AOSE) methodologies that describe the process of converting a set of requirements to MAS design artifacts. The AOSE methodologies have devised techniques and notations to describe the interaction among agents that can be used to derive the behavioral description of the MAS. Every AOSE methodology includes some kind of model that describes the interaction among agent types or classes that constitute the agents. One of the approaches to describe such interactions is using scenarios. A scenario is a temporal sequence of interaction events (i.e. messages) among agents. Using scenarios, designers describe system's functionality using partial and local interactions of the constituent agents. The scenarios can be used to verify the MAS design. There are two main ways of representing scenarios, namely, OMG's Sequence Diagram (SD) (also known as event scenarios and event trace diagrams) [1] and ITU's Message Sequence Chart (MSC) [2]. In this paper we use the latter.

There are several advantages of using scenarios such as expressive power and simplicity. However, there are also several challenges such as weak partial ordering semantics that may lead to missing some behavioral requirements particularly for MAS with distribution of control. For instance, because each scenario gives a local and partial story of interaction between two or more agents, the challenge is how the behavior of the whole MAS can be constructed from the scenarios, and more importantly, whether the derived behavior is acceptable or not.

The model which describes the behavior of each agent is usually called *behavioral model*, and the procedure of building the behavioral model from a scenario-based specification, is called *synthesis of behavioral models*, or simply, *synthesis process*. A commonly used model for behavioral modeling of individual agents is the state machine. There are several reports on the procedure of converting a set of scenarios to a behavioral model expressed by state machines [3-9]. In the synthesis process, one state machine will be built for each agent. The state machine includes all the interactions of a particular agent based on the messages that it receives or sends. Theoretically, the behavior of the MAS can be described by the union (parallel execution) of all the state machines of the individual agents.

One of the challenges during the synthesis process, is *implied scenarios* [10-13], also known as *emergent behavior*. An implied scenario is a specification of behavior that is in the synthesized model of the MAS but is not explicitly specified in the set of scenarios. This usually happens when several autonomous agents need to handle a joint task as a group in a shared environment where control is also distributed. In this paper we use an example of MAS for semantic search to demonstrate the ideas.

In real world applications, detecting implied scenarios during the design phase is highly appreciated. So far there is no formal representation for the cause of implied scenarios, so that by removing that cause during the design phase, the emergent behavior can be eliminated. *Safe realizability* is a measure that shows the behaviors specified in the specification. In [11] an algorithm for safe realizability is proposed but there is no method to check whether implied scenarios exist in the first place. A main contribution of this paper is to define indeterminism in behavior of multi-agent systems, so that by identifying indeterminism, one can detect the potential emergent behavior.

The structure of the paper is as follows: in Section 2 the multi-agent system for semantic search that is used as an example throughout this paper is introduced. In Section 3 system behavioural modeling is explained using the semantic search system. Detection of indeterminism and emergent behaviour is discussed in Section 4. Section 5 contains the approach for validation of MAS against a particular unwanted behaviour. The design validation software is presented in section 6 and conclusions are drawn in section 7.

2. MAS for Semantic Search

Traditional search engines depend on the number of occurrence of words in document. Semantic search depends on understanding the meaning of the concepts used in the context of other words. It then tries to retrieve the related documents to these concepts. The backbone of semantic search is the semantic interoperability which is the main ingredient for notation extraction from the search phrase.

2.1. Semantic Search Process

We have devised a spiral workflow to incorporate both search and concept learning in the semantic search process [14]. The spiral workflow and its suggested scenario are shown in Figure 1.

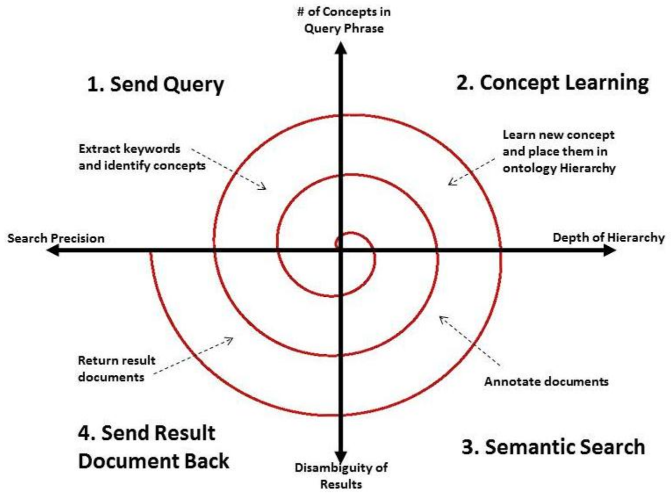

# of Concepts in Query Phrase
1. Send Query
2. Concept Learning
Extract keywords and identify concepts
Learn new concept and place them in ontology Hierarchy
Search Precision
Depth of Hierarchy
Return result documents
Annotate documents
4. Send Result Document Back
Disambiguity of Results
3. Semantic Search

Figure 1 - Spiral Workflow between Semantic Search and Concept Learning

The scenario of this process is as follow:

- The system is initiated by the user when he/she sends the search query to one agent (we call it local agent).
- Concepts are extracted from the search query.
- The concepts in the search query are compared to those defined with the ontology in the local repository. This comparison is essential to enable the local agent to find out the new concepts in the search query that are not defined in its local repository.
- If new concepts are found in the search query, the local agent requests the other agents (remote agents) to teach it these new concepts. (This step represents the initialization of the concept learning process.)
- The concept learning mechanism is continued until the local agent learns the new concepts adequately.
- After learning all new concepts, the concept hierarchy in local repository is reorganized to add the new concepts in their proper position.
- The annotation procedure is then performed on the fly on all the concepts in the local repository; old and newly learned concepts. UIMA [15] is used to enable search and classification within each document repository.
- In the same time, the local agent broadcasts the search query to all remote agents to search their local repository and send back the result documents.

The local agent collects the returned documents from remote agents and ranks them using social networks before retrieving them to the user.

2.2 Semantic Search Infrastructure

In the previous section, we described the importance of the concept learning module in the semantic search system. The main obstacle in the concept learning system is the complexity of ontological heterogeneity solution. In order to solve this problem we propose to leverage of the power of a multi-agent system to use it in the infrastructure of our semantic search system. MAS are used to set up the backbone elements that communicate with each other. As shown in Figure 2, each MAS controls a repository that uses an ontology to cover a specific area of knowledge. Each repository contains some documents that are related to concepts defined within the ontology. Using MAS allows the system to hide the search complexity from the user.

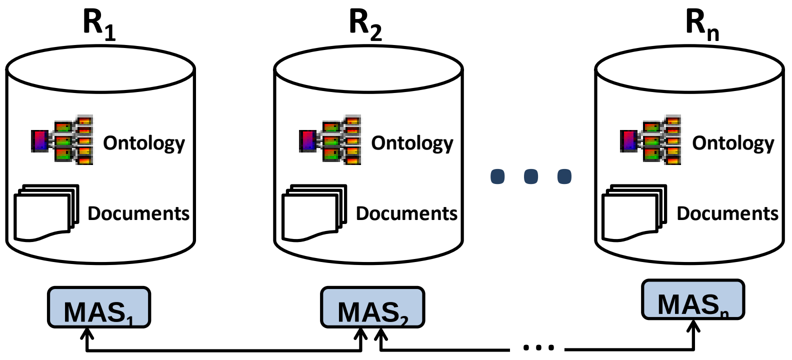


Figure 2 - The System Architecture

2.3 Prototype System Architecture

Figure 3 shows different agent roles in each MAS in our prototype system. These roles are defined below.

Query Handler: This role involves accepting search query and processing it by extracting concepts from it. Also it is responsible for broadcasting the query statement to all the neighbor repositories.

Concept Manager: This role involves finding the new concepts in the search query and broadcasting it to all neighbour agents in order to be learnt.

Concept Learner: This role involves maintaining and confirming newly learnt concepts; including creation of taxonomies of interested domain. It is also responsible for broadcasting these concepts to all group members to share searching for it. Moreover it rearranges local repositories with the newly learned concepts.

Document Annotator: This role involves annotating the documents in local repository and filtering them according to the search keywords, then returning back the filtered documents.

Peer Finder: This role involves detecting cooperative peers (agents) that communicate with the current agent with a relationship.

Tie Manager: This roles involves keeping track of common concepts between peers and the interactions occur between those peers in the learning process to be able to change the strength of tie between them dynamically. It is also responsible for setting the initial strength of the relationship between agents.

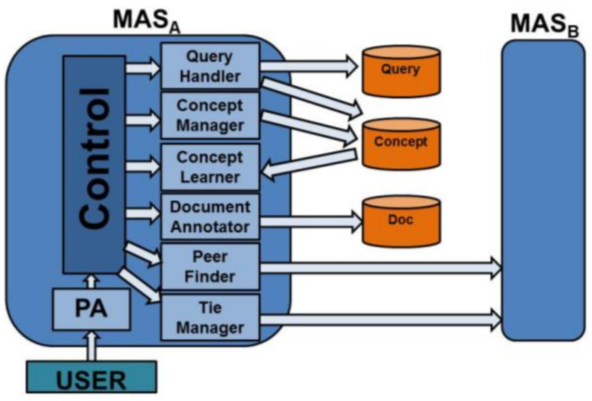


Figure 3 - Roles of Different Agents

3. System Behavioral Modeling

The requirements for the semantic search MAS presented in Section 2 are captured by the partial message sequence charts (pMSCs). The pMSCs will be formally defined in Definition 1, below. The message sequence chart 1 (MSC1), shown in Figure 4, illustrates a scenario where upon entering the query by the user, the query handler (QH) agent extracts the concepts from the query and sends it to the concept learner (CL) agent and the local repository (REP). In this scenario the CL does not detect any new concepts among the extracted concepts which may require learning, and the results of the search are returned by the local repository. MSC2 however (shown in Figure 5), illustrates a scenario in which upon entering the query by the user and extracting concepts by the QH, the CL detects new concepts which it needs to learn. Thus the learning process commences; the local repository is updated, and the results are sent back to the user. MSC3, shown in Figure 6, exemplifies an emergent behaviour that could be derived from MSC1 and MSC2. In this scenario, it is shown that there could emerge a case, where the concept learner detects concepts that need learning and starts the learning process, however before CL has the chance to update the local repository, the results are sent back to the user by the local repository. Therefore the user will not be provided with the best and the most up-to-date results. The problem is how one can detect such a scenario (MSC3) given MSC1 and MSC2. This can be done in an ad-hoc way [9] or a systematic way using our proposed semantic causality notion [16, 17] (see Section 4).

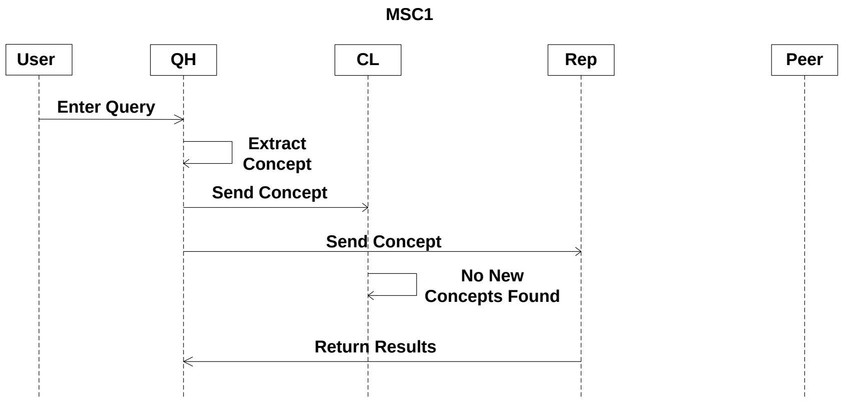


Figure 4 - Concept learner does not detect any new concepts that need to be learnt. Results from the local repository are sent to the user

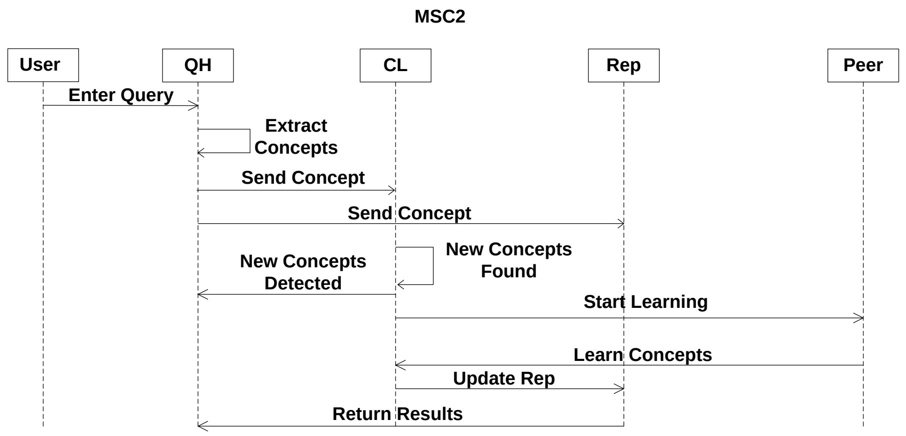


Figure 5 - Concept learner detects new concept that needs to be learnt. Learning process begins and the local repository is updated. Results from the local repository are sent to the user

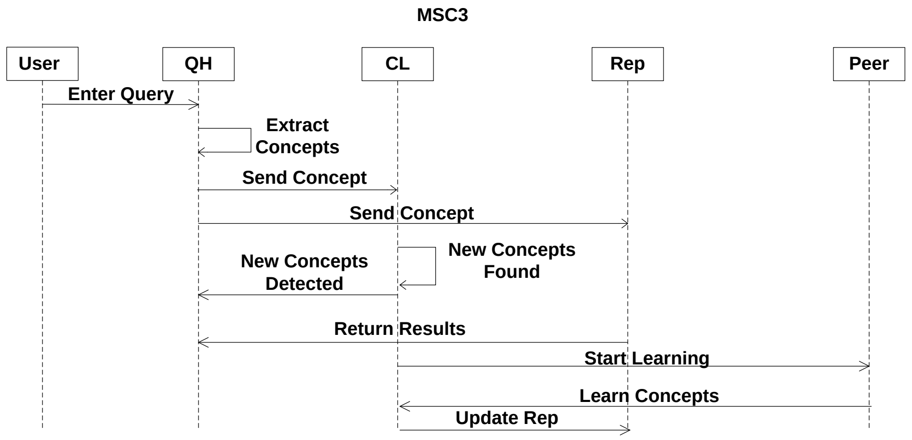


Figure 6 - Repository returns search results to QH before it has been updated by CL

3.1 Definitions

In this section, we give some definitions related to MSC notation based on a subset of ITU definitions for MSC [2, 10, 18, 19].

Let P be a finite set of agents in a distributed system (with the total number of agents $p \geq 2$) and C be a finite set of message contents (or message labels) that are passed among the agents. Let $\sum_i = \{i!j(c), i?j(c) | j \in P\backslash\{i\}, \; c \in C\}$ be the set of alphabet (i.e. events) for the agent $i \in P$, where $i!j(c)$ denotes an event that sends a message from agent i with content c to agent j, whereas $i?j(c)$ denotes an event that is received by agent i a message with content c from agent j. The set of alphabet will be $\Sigma = \bigcup_{i \in P} \Sigma_i$ and each member of Σ is called a message.

In the following, we try to capture a causal relationship between a message and its predecessors by defining partial Message Sequence Chart (pMSC).

Definition 1. (partial Message Sequence Chart): A partial Message Sequence Chart (pMSC) over P and C is defined to be a tuple $m = (E, \alpha, \beta, \prec)$ where:
E is a finite set of events.
$\alpha: E \rightarrow \Sigma$ maps each event with its label. The set of events located on agent i is $E_i = \alpha^{-1}(\Sigma_i)$. The set of all send events in the event set E is denoted by $E! = \{e \in E \mid \exists i, j \in P, c \in C : \alpha(e) = i!j(c)\}$ and the set of receive events as $E? = E \backslash E!$.
$\beta: F! \rightarrow E?$ is a bijection mapping between send and receive events such that whenever $\beta(e_1) = e_2$ and $\alpha(e_1) = i!j(c)$, then $\alpha(e_2) = j?i(c)$.
$\prec$ is a partial order on E such that for every agent $i \in P$, the result of $\prec$ on E_i is a total order of its members and the transitive closure of $\{(e_1, e_2) | e_1 \prec e_2, \exists i \in P: e_1, e_2 \in E_i\} \cup \{(e, \beta(e)) | e \in E\}$ is a partial order of the members of E.

The partial order $\prec$ captures causal relationship between the events of a pMSC. This causality basically represents two things. First, a receive event cannot happen without having its corresponding send event happened before. Second, a receive (or send) event, cannot happen until all the previous events, which are causal predecessors of it have already been accomplished. Obviously, if all the send events have their corresponding receive events (i.e. as defined by the function β), the structure is called a Message Sequence Chart or simply an MSC. In other words, an MSC has the same structural components as a pMSC, except that β is defined for $F! = E!$.

Definition 2. (projection): The projection $m|_i$ for agent i in MSC m, is the ordered sequence of messages corresponding to the events for the agent i in the pMSC m. For $m|_i$, $\|m|_i\|$ indicates its length, which is equal to the total number of events of *m* for the agent *i*, and $m|_i[j]$ refers to *j*th element of $m|_i$, so that if e_j is the *j*th interaction event for agent *i* according to the total order of the events of *i* in *m*, then $\alpha_m(e_j) = m|_i[j-1], 0 \prec j \prec \|m|_i\|$. In $m|_i$, we call every element $i!j(c), i, j \in P, c \in C$, a send message and every element $i?j(c)$, a receive message.
For example, the projection for the agent QH in MSC1 in Figure 2 will be "QH!CL(send concept) ".

Definition 3. (Equivalent Finite State Machine for a projection): For the projection $m|_i$, we define the corresponding deterministic finite state machine $A_i^m = (S^m, \Sigma^m, \delta^m, q_0^m, q_f^m)$ such that:
S^m is a finite set of states labelled by q_0^m to $q_{\|m|_i\|}^m$.
Σ^m is the set of alphabet
q_0^m is the initial state
$q_f^m = q_{\|m|_i\|}^m$ is the final state (accepting state)
δ^m is the transition function for A_i^m such that $\delta(q_j^m, m|_i[j]) = q_{j+1}^m, 0 \leq j \leq \|m|_i\| - 1$. Thus the only word accepted by A_i^m is $m|_i$.

Note that scenarios can be treated as *words* in a formal language, which is defined over send and receive events in MSCs. Then, a *well-formed word* for an agent is one that for every receive event there exists a send event in that word, which in fact captures the essence of definition given for a pMSC (Definition 1). On the other hand, a *complete word* for an agent is the one that for every send event in it, its corresponding receive event also exists in it. In practice, a system designer must look for complete and well-formed words for each agent, which is not necessarily an easy task. For any MSC *m* in the set of MSCs *M*, any sequence *ω* of *m*, obtained from a sequence of events in *m* that respects the partial order of the events defined for *m*, is called a linearization of *m*, and is a word in the language *L(M)* of *M*.

3.2 Constructing Behavioural Model

As mentioned in the previous section, scenario based specification is an efficient and effective way to represent system requirements for multi-agent systems. However as each scenario only partially describes system's behaviour, scenario based specifications are subject to deficiencies such as incompleteness and contradictions. Therefore having a methodology which can systematically discover system design errors before implementation will result in enormous savings in time and cost. The first part of this approach, which is the synthesis of state machines from message sequence charts is described in this section and is demonstrated using the example of the semantic search system.

The first step is to construct behaviour models for individual agents using finite state machines (FSMs). As explained earlier, the procedure of constructing FSMs from message sequence charts (MSCs) is referred to as behavior modeling. For any agent *i* of a partial message sequence chart (pMSC) defined in Definition 1, an equivalent finite state machine (Definition 3) can be constructed. For the example of semantic search system, behavior model of the query handler (QH) agent is demonstrated. Figures 7 and 8 show the eFSMs that are constructed for the QH agent in MSC1 (Figure 4) and MSC2 (Figure 5) respectively.

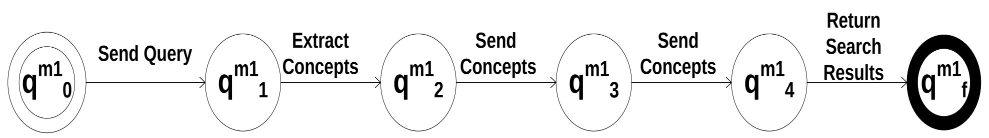


Figure 7 - eFSM for the QH agent in MSC1

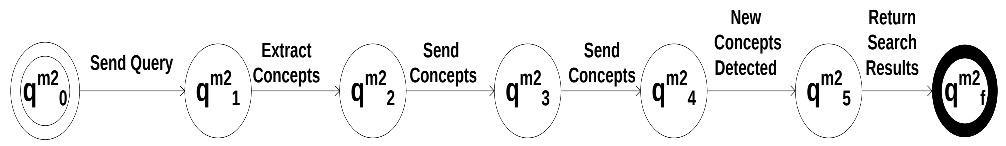


Figure 8 - eFSM for the QH agent in MSC2

To complete behavior model for the QH agent another FSM, which is the union of its corresponding eFSMs from different scenarios that the QH participates in, is built and shown in Figure 9.

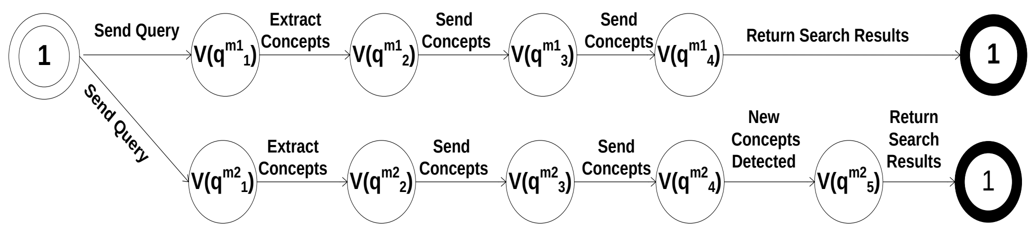


Figure 9 - The union of the eFSMs built from MSCs 1 and 2

4. Detection of Emergent Behavior

Emergent behaviour occurs when there exists a state, in which the agent becomes confused as to what course of action to take. This happens when identical states exist in the union of eFSMs obtained through behavioural modelling. A definition for identical states is needed for detection of emergent behavior. To achieve this we must first have a clear procedure to assign values to the states of the eFSMs. This is a very important step and is performed differently in various works. For instance, [9] proposes the assignment of global variables to the states of eFSMs by the system designed engineer (referred to as the domain expert in this research). However the outcome of this approach is not always consistent as the global variables chosen by different domain experts may vary. Therefore to achieve consistency in assigning state values, the approach of [14] which is making use of an invariant property of the system called semantic causality is followed.

Definition 4. (Semantic causality): A message $m|_i[j]$ is a semantical cause for message $m|_i[k]$ and is denoted by $m|_i[j] \overset{se}{\rightarrow} m|_i[k]$, if agent i has to keep the result of the operation of $m|_i[j]$ in order to perform $m|_i[k]$.
For example, in MSC1 in Figure 2, message "extract concepts" is a semantic cause for message "send concepts". As semantic causality is an invariant property of the system and is part of the system's architecture and the domain knowledge, it is independent of the choices made by the domain experts. In other words, we let the current state of the agent to be defined by the messages that the agent needs in order to perform the messages that come after its current states. Thus in order to evaluate state values of the resulting FSM, a domain theory which consists of the domain knowledge of the system must be constructed as defined formally in Definition 5.

Definition 5. (Domain theory): The domain theory D_i for a set of MSCs M and agent $i \in P$ is defined such that for all $m \in M$, if $m|_i[j] \overset{se}{\rightarrow} m|_i[k]$ then $(m|_i[j],\ m|_i[k]) \in D_i$.
Continuing with the above example, since the message "extract concepts" is a semantic cause for message "send concepts", both messages are part of the domain theory. However building the domain theory can be very time consuming. Therefore as a part of this systematic approach, building a light domain theory is introduced. The concept of light domain theory is closely tied to the calculated state values as defined in Definition 6. Using this definition, it becomes evident that only states with the same incoming transitions have the potential to exhibit indeterministic behaviour which have the same incoming transitions. Assigning state values to states of eFSMs is done by making use of semantic causality as defined in Definition 6.

Definition 6. (State value): The state value $v_i|(q_k^m)$ for the state q_k^m in eFSM $A_i^m = (S^m, \Sigma^m, \delta^m, q_0^m, q_f^m)$ is a word over the alphabet $\Sigma_i \cup \{1\}$ such that $v_i|(q_f^m) = m\ |_i\ [f - 1]$, and for 0 < k < f is defined as follows:

i) $v_i|(q_k^m) = m\ |_i\ [k - 1] v_i|(q_j^m)$, if there exist some j and l such that j is the maximum index that $m|_i[j - 1] \overset{se}{\rightarrow} m|_i[l], 0 < j < k,\ k \leq l < f$

ii) $v_i|(q_k^m) = m\ |_i\ [k - 1]$ if case i) does not hold but $m|_i[k - 1] \overset{se}{\rightarrow} m|_i[l]$, for some k ≤ l < f

iii) $v_i|(q_k^m) = 1$, if none of the above cases hold

For instance in order to calculate the state value for state q_4^{m1} we proceed as follows: from the domain theory of the system (Definition 5) we learn that the maximum index j for which $m1\ |_{QH}\ [j - 1]$ is a semantical cause for a message in the transitions after q_4^{m1} is j = 2 for which $m1\ |_{QH}\ [j - 1] = ExtractConcepts$. That is to say that for example the message "Extract Concepts" is a semantic cause for message "Send Concepts". Therefore from case (i) of Definition 6 we obtain: $v_{QH}|(q_4^{m1}) = m\ |_{QH}\ [2 - 1] v_{QH}|(q_2^{m1})$.

In order to calculate $v_{QH}|(q_2^{m1})$ we observe that "extract concepts" is the only semantic cause after q_2^{m1}, thus case (ii) of Definition 6 holds and we get $v_{QH}|(q_2^{m1}) = Extract\,Concepts$. Therefore we have $v_{QH}|(q_4^{m1})$ = *(Send concepts)(Extract Concepts)*. By following the same approach we get the state value for q_4^{m2} to be $v_{QH}|(q_4^{m2})$ = *(Send concepts)(Extract Concepts)*.
From these examples it can clearly be seen that semantic causality is an invariant property of the system and is not affected by the preferences of the domain expert.
Getting back to our systematic approach to finding emergent behaviour, by considering the resulting FSM in Figure 9, we select pairs of states with the same incoming transitions and evaluate their state to look for identical states. Figure 10 illustrates the constructed FSM as a result of the merging of identical states.

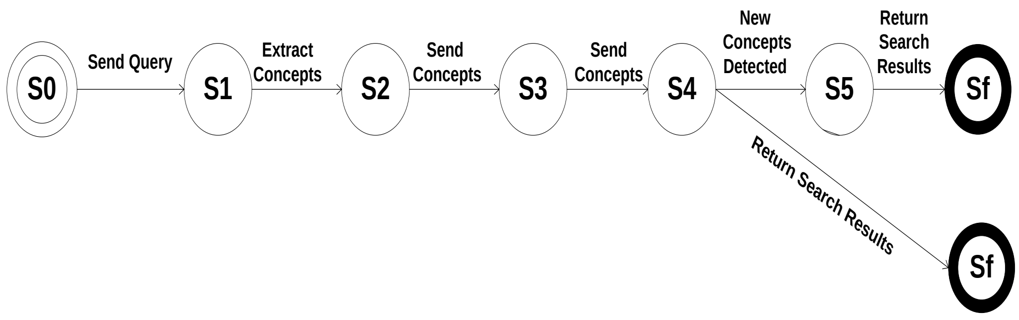


Figure 10 - Resulted FSM after Merging Identical states

As it is shown, state S4 is where QH agent falls into confusion. That is, as it is illustrated in MSC3 (Figure 6), QH will not be able to distinguish whether or not it is getting the most updated search results. In other words, it is entirely possible that QH is notified that there are new concepts found in the query by the CL after it has already notified the user with the search results. Therefore as a result of the systematic approach in detecting emergent behavior in MAS, the domain expert is notified of such possible scenarios and is able to make modifications where necessary.

5. System Validation

As it was shown in the previous sections emergent behaviors can occur in MAS due to incomplete and faulty requirements. Section 4 demonstrated methodologies to detect emergent behaviors in MAS. However it is often desirable that the lack of certain behaviors in MAS is ensured and validated. In this approach system engineers will have a set of undesired scenarios and the system requirements (which are also expressed using scenarios) are checked to verify that the illegal scenarios from this set cannot be derived from them.
Below we present a methodology which takes two different sets of scenarios expressed using MSCs as follows:

A. A set of MSCs containing scenarios describing the system's legal behaviour

B. A set of illegal scenarios which are undesirable to occur

By continuing with our case study of semantic search engine, set A is made up of MSCs 1 and 2 (Figures 4 and 5). A possible illegal scenario which is undesired to occur is illustrated in Figure 11. The scenario demonstrated in this Figure 11 shows a case where the search results are returned to QH without considering new concepts.

By having the two sets of scenarios (Set A and B) which this methodology attempts to verify that scenarios in set B cannot be derived from scenarios of set A. In other words this methodology ensures that system's behavior does not contain scenarios from set B. This methodology is divided into two parts of constructing the behavioral model and ensuring the lack of invalid scenarios in the built models. The procedure of building behavioral models was covered extensively in section 3 and in fact we will make use of the model built in that section (shown in Figure 9) for the purposes of this approach. It is important to note that since behavioral modeling is the basis for both approaches presented in section 4 and this section, they can be executed very efficiently. We only need to build new behavioral models for scenarios in set B.

Upon the identification of cases of emergent behavior, a new set C can be constructed to contain their related behavioral models. In other words, set C contains all cases of emergent behavior in the system. Therefore if a behavior model built based on scenarios in set B does not match a behavioral model in set C, it is verified that the system will not contain that particular illegal scenario. Conversely if a behavior model constructed based on the scenarios of set B is equal to the behavioral model in set C, the verification has failed. By comparing the FSM in Figure 9 with the behavior model constructed from set B (Figure 8) it becomes evident that the illegal scenario of set B can potentially emerge from the scenarios of set A.

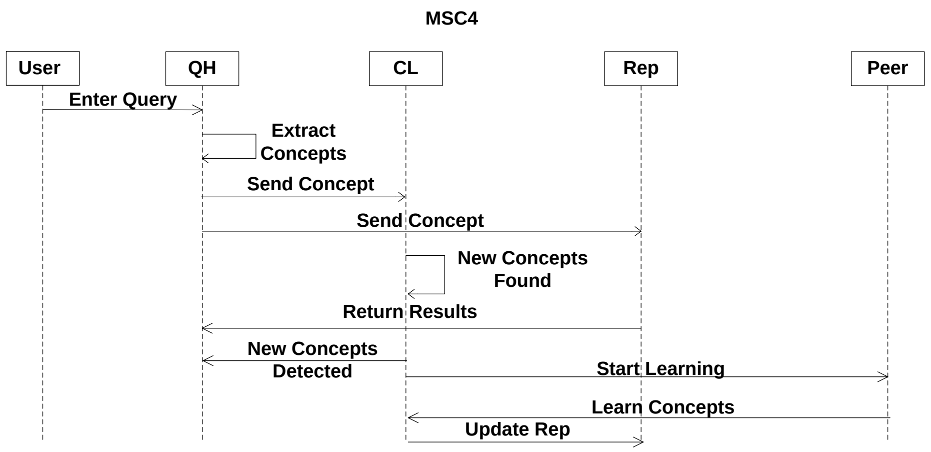
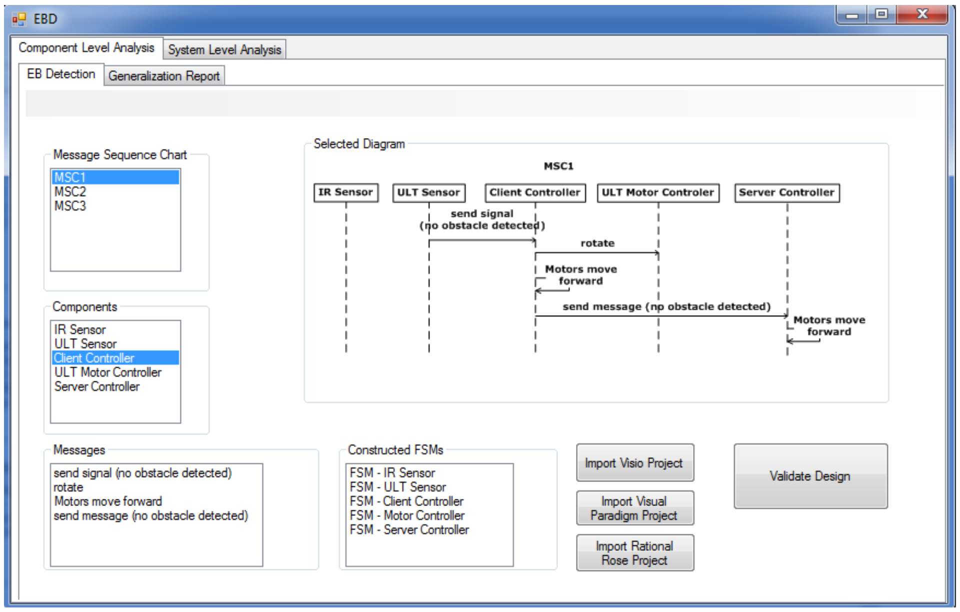


Figure 11 - Repository returns search results to QH without considering new concepts

6. Design Validation Tool

As presented in the preceding sections of this paper, despite many advantages of using scenario-based specification for the design of MAS, there are certain limitations to this approach. Therefore in order to use scenarios, it is greatly beneficial and even necessary to devise methodologies which verify the resulted design of the system. Consequently in order to enable the efficient and effective use of these methodologies in real world projects, they need to be made automated in a user-friendly software package. In this section, the design and implementation of this tool is presented.

The second concern was that since this tool is developed to increase the efficiency of the development life cycle, it is highly desirable that it is easy to use. To incorporate the usability of the tool, an easy to follow graphical user interface has been developed (Figure 12) which closely represents the logical flow of the developed methodology. Moreover, as MSCs are constructed via a variety of different software packages, in order to account for the convenience of the users, this tool can import MSCs from IBM Rational Rose and Microsoft Office Visio.



Message Sequence Chart: MSC1, MSC2, MSC3

Components: IR Sensor, ULT Sensor, Client Controller, ULT Motor Controller, Server Controller

Messages:
send signal (no obstacle detected)
rotate
Motors move forward
send message (no obstacle detected)

Constructed FSMs:
FSM - IR Sensor
FSM - ULT Sensor
FSM - Client Controller
FSM - Motor Controller
FSM - Server Controller

Figure 12 - Snapshot of the GUI of the tool; displaying an imported MSC

In the development of this tool, there were two major concerns. First as there is still a vast amount of research remaining in this area, as outlined in the conclusions and future work section of this paper, it is essential for this tool to be modular and scalable. To comply with this requirement this tool was built on the two pillars of encapsulation and parallel execution.

As it can be seen from the snapshot of the tool's graphical user interface shown in Figure 12, upon importing a design project from one of the above mentioned tools, the data boxes of the GUI are populated automatically with appropriate data. By clicking any of the imported MSCs, the components of that MSC will be shown in the *Component* subsection of the GUI and the actual MSC will be shown in the *Selected Diagram* area. Consequently, by selecting a component, the messages associated with that component will be shown in the *Message* subsection of the GUI. The synthesis of the behavior model, explained in section 4 of this paper is conducted immediately upon importing related MSCs. As a result the built FSMs will be shown in the *Constructed FSMs* subsection of the GUI. By clicking on the title of any of the FSMs the constructed figure will be shown in the Selected Diagram area as shown in Figure 13.

At this point, by clicking the Validate Design button, the methodology commences. The user will be asked to assist in constructing the domain theory which will be used to find identical states as outlined in section 5 of this paper. Upon completion of the analysis, the user will be presented with a report outlining the areas in which indeterminism could occur.

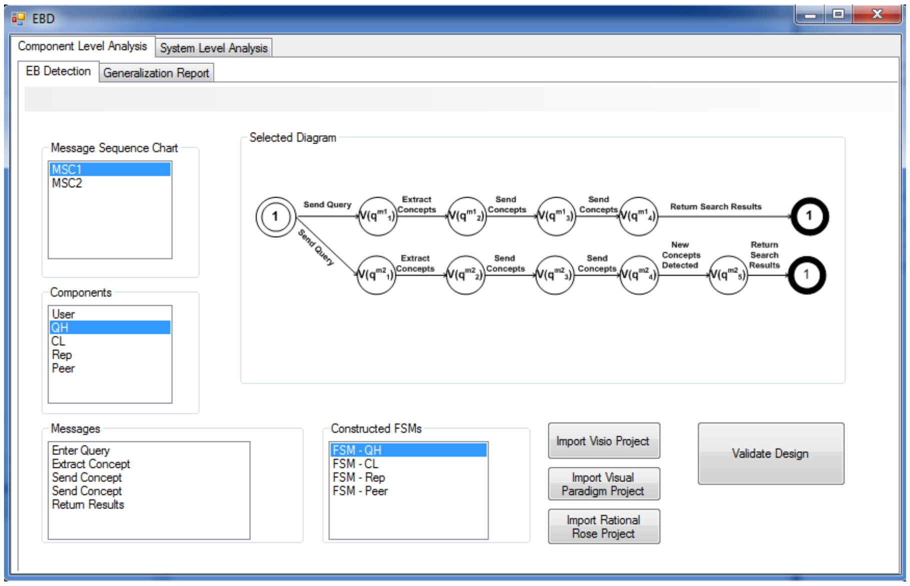


Figure 13 - Snapshot of the GUI of the tool; displaying a constructed FSM

7. Conclusions and Future Work

Some of the failures in distributed systems can be directly attributed to their design. Unfortunately, manual review of the design documents may not efficiently detect all the design flaws due to the scale and complexity of the system. Detecting unwanted behavior during the design phase is about 20 times cheaper than finding them during the deployment phase [20]. In this paper we devised and demonstrated a method to detect and remove design flaws of a MAS that may lead to emergent behaviour.

This method is novel in the sense of formalization of the cause of implied scenarios. We believe that this is the main reason for some shortcomings and conflicts in the current works, as they have been revealed in [21] and [17]. One of our contributions is to show that the concept of indeterminism, which was demonstrated in this paper, is able to address and resolve the problems that are mentioned in [21]. The presented software provides an easy to use and practical tool to apply these algorithms to requirements and design documents of distributed systems.

For future work, extending the proposed methodology to a comprehensive framework for model-based analysis and testing of distributed software systems is intended. Furthermore, this technique can be modified to take the UML's sequence diagrams as input and thus incorporate the analysis and design of distributed object oriented systems.

References

[1] "Unified Modeling Language Specification. Version 2. Available from Rational Software Corporation," Cupertino, CA, 2006.

[2] "Recommendation Z.120: Message Sequence Chart(MSC)," Geneva, 1996.

[3] D. Harel and H. Kugler, "Synthesizing state-based object systems from lsc specifications," *International Journal of Foundations of Computer Science*, 2002.

[4] I. Kruger, R. Grosu, P. Scholz, and M. Broy, "From mscs to statecharts," in *Franz j. rammig (ed.): Distributed and parallel embedded systems*: Kluwer Academic Publis, 1999.

[5] E. Makinen and T. Systa, "MAS - an interactive synthesizer to support behavioral modeling in UML," in *ICSE 2001* Toronto, 2001.

[6] M. Moshirpour, "Model-Based Detection of Emergent Behavior In Distributed and Multi-Agent Systems from Component Level Perspective," MSc Thesis, *Department of Electrical and Computer Engineering*, University of Calgary, 2011.

[7] M. Moshirpour, A. Mousavi, and B.H. Far, "Detecting Emergent Behavior in Distributed Systems Using Scenario-Based Specifications," in *Proceedings of the International Conference on Software Engineering and Knowledge Engineering* San Francisco Bay, USA, 2010.

[8] S. Uchitel, J. Kramer, and J. Magee, "Synthesis of behavioral models from scenarios," *IEEE Transaction on Software Engineering,* pp. 99-115, February 2003.

[9] J. Whittle and J. Schumann, "Generating statecharts designs from scenarios," in *ICSE* Limerick, Ireland, 2000.

[10] B. Adsul, M. Mukund, K.N. Kumar, and V. Narayanan, "Casual Closure for MSC Languages " in *FSTTCS*, 2005, pp. 335-347.

[11] R. Alur, K. Etessami, and M. Yannakakis, "Inference of Message Sequence Charts," *IEEE Transaction on Software Engineering,* pp. 623-633, July 2003.

[12] H. Muccini, "Detecting implied scenarios analyzing nonlocal branching choices," in *FASE 2003* Warsaw, Poland.

[13] S. Uchitel, J. Kramer, and J. Magee, "Negative scenarios for implied scenario elicitation," in *10th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE 2002)* Charleston.

[14] B.H. Far, C. Zhong, Z. Yang, and M. Afsharchi, "Realization of Semantic Search Using Concept Learning and Document Annotation Agents," in *Proceeding of Twenty-First International Conference on Software Engineering and Knowledge Engineering (SEKE)*, 2009, pp. 164-169.

[15] A. Lally, K. Verspoor, and E. Nyberg, "Unstructured Information Management Architecture (UIMA) Version 1.0, (OASIS, 2008.)."

[16] M. Moshirpour, A. Mousavi, and B.H. Far, "A Technique and Tool to Detect Emergent Behavior of Distributed Systems Using Scenario-Based Specifications," in *Proceedings of the International Conference on Tools with Artificial Intelligence*, Arras, France, 2010.

[17] A. Mousavi and B.H. Far, "Eliciting Scenarios from Scenarios," in *Proceedings of 20th International Conference on Software Engineering and Knowledge Engineering (SEKE 2008)* San Francisco Bay, USA: July 1-3, 2008, 2008.

[18] "ITU: Message Sequence Charts. Recommendation, International Telecommunication Union. ," 1992.

[19] M. Lohrey, "Safe realizability of high-level message sequence charts," *CONCUR,* vol. 177-192, 2002.

[20] D. R. Goldenson and D. L. Gibson, "Demonstrating the Impact and Benefits of CMMI: An Update and Preliminary Results," *CMU/SEI-2003-SR-009,* October 2003.

[21] A.J. Mooij, N. Goga, and J.M.T. Romijn, "Non-local choice and beyond: Intericacies of MSC choice nodes," in *M. Cerioli (ed): FASE 2005, LNCS 3442*: Springer, 2005, pp. 273-288.

Authors

Mohammad Moshirpour received two BSc. degrees in Computer Science and Software Engineering from the University of Calgary, Canada, in 2008 and 2009 respectively. He received his MSc. in Software Engineering from the University of Calgary in 2011. Currently Mohammad is a PhD. student at the Department of Electrical and Computer Engineering in the University of Calgary. His Research interests include automated analysis of design and requirements of distributed and multi-agent systems.

Shimaa M. El-Sherif received BSc. and MSc. degrees from Electrical and Computer Engineering department Shoubra Faculty of Engineering (SFE) Benha University Egypt and is currently pursuing a PhD. in Software Engineering in department of Electrical and Computer Engineering University of Calgary, Canada. During 2001-02 she worked as a software developer for NewSoft, Egypt and during 2002-08 she has worked as an instructor and research assistant for SFE. Her current research interests include: Multi-Agent systems, semantic search, concept learning from multiple teachers with diverse ontologies and using social networking in both concept learning and semantic search to improve the learning and search results.

Abdolmajid Mousavi received his MSc. in electrical and computer engineering from Tehran University, Iran, in 1997, and his PhD. from the Department of Electrical and Computer Engineering, University of Calgary, in 2009. From 1997 to 2003, he was jointly the manager of the computer networks division of Behineh Pardazi Co., in Iran, a lecturer at the University of Lorestan, Iran, and a consultant for medical instrumentation standards in the National Institute of Industrial Research and Standardization of Iran. His main research interests are software verification, application of artificial intelligence, and formal methods. Currently, he is the head of computer engineering group at the University of Lorestan, Iran.

Behrouz H. Far received the BSc. and MSc. degrees in Electrical Engineering from Tehran University in 1983 and 1986 respectively, and PhD. from Chiba University, Japan, in 1990. Before moving to the University of Calgary, where he is an Associate Professor at the Department of Electrical and Computer Engineering, he was an Associate Professor at the Department of Information and Computer Sciences, Saitama University. His research interests are model-driven development, analysis and testing of distributed and multi-agent systems.