Model Based Approach to Detect Emergent Behavior in Multi-Agent Systems [Extended Abstract]

Mohammad Moshirpour[1], Nariman Mani[2], Armin Eberlein[3], Behrouz H. Far[1]
[1]University of Calgary, Department of Electrical and Computer Engineering, 2500 University Dr NW, Calgary, AB, Canada, T2N 1N4
[2]Carleton University, Department of Systems and Computer Engineering, 1125 Colonel By Drive Ottawa, Ontario, Canada, K1S 5B6
[3]American University of Sharjah, Department of Computer Science & Engineering, UAE
{mmoshirp, eberlein, far}@ucalgary.ca
{nmani}@sce.carleton.ca

ABSTRACT

Multi-agent systems (MAS) are efficient solutions for commercial applications such as robotics, business commerce applications, information retrieval and search engines. In 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 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. Effective and efficient design validation of MAS requires the development of systematic and automated methodologies to review MAS design documents. Although the increasing demand for MAS in the software industry has led to the development of several Agent Oriented Software Engineering (AOSE) methodologies, the AOSE methodologies usually do not fully cover monitoring and testing. In this paper, a technique to help MAS developers verify, test and monitor MAS design is introduced. This method uses MAS analysis and design artifacts created by the MaSE AOSE methodology. In this technique, design artifacts of MaSE are converted to scenario-based specification, which is very similar to UML's sequence diagrams. Then the specifications are used to analyze the system for validating the design of MAS and ensuring the lack of emergent behavior.

Categories and Subject Descriptors
D.2.4 Software/Program Verification

General Terms
Design, Verification

Keywords
Distributed systems, Multi-agent systems, Emergent behavior, Verification, Software testing, Message sequence chart, State machine, Certification, Scenario based specification.

Appears in: *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013),* Ito, Jonker, Gini, and Shehory (eds.), May, 6–10, 2013, Saint Paul, Minnesota, USA.

1. INTRODUCTION

In software engineering, a practical approach for software development is describing system requirements using scenarios [1] (including variations aka. user stories [2], UML's sequence diagrams [3] and message sequence charts [4]). A scenario is a temporal sequence of interaction events (i.e. messages) among system components (processes, actors or agents). Scenarios describe how system components work concurrently and interact in order to provide system level functionality. In a scenario the assumptions on how time is represented determine a partial ordering of events [4]. Each scenario is like a window where only a part of interaction among components is visible. A component may participate in several scenarios. Consequently, a mechanism to represent the behavior of each component as a single entity is needed. This is called behavior modeling and is usually represented by a state machine [3]. The system behavior, in turn, will be the concurrent execution of the components' state machines. Although handy and expressive, scenario-based specification and behavioral modeling based on it are prone to subtle drawbacks including incompleteness and partial description.

The main concern of this research is the emergent behavior that can arise when behavior model of a system with multiple components is synthesized from its scenario-based specification. The emergent behavior can be analyzed and detected in two different ways: (1) focusing on a component's behavior, emergent behavior may arise from the merger of partial specification that the component participates in; (2) focusing on the system behavior, emergent behavior may arise depending on assumptions and the technique used in the behavior synthesis process. In this work the artifacts produced by the Multi-agent Software Engineering (MaSE) methodology [5] are used. MaSE is a detailed and popular approach for the analysis and design of MAS. This methodology utilized several diagrams and models resembling the standard Unified Modeling Language (UML) to describe the architecture-independent structure of agents and their interactions.

A contribution of this paper is devising an algorithm to extract scenarios that make up the overall functionality and behavior of the MAS from MaSE models. Having access to the scenario-based specifications of MAS is considered greatly valuable as not only scenarios are an efficient way to describe system's requirements and behavior, but they can also be used to examine the system for possible design faults such as emergent behavior. In this research a systematic approach is proposed to extract scenarios from MaSE analysis and design artifacts. These scenarios are then used to examine the design of the MAS. In this paper, a software system to verify the design of MAS by employing the proposed technique is also presented.

2. METHODOLOGY

Among the MaSE models [5], this research uses the sequence diagrams in the "Applying Use Cases" step from the analysis phase of MaSE along with the "Agent Class Diagrams" from the design phase of MaSE to construct the partial scenarios. This is since the role sequence diagrams of the "Applying Use Cases" step of MaSE contain the conversations among roles assigned to each agent [5].

2.1 Extracting Scenarios from MaSE Models

The above mentioned MaSE models are then used to construct scenarios for the system under construction. The approach for extracting scenarios in the form of message sequined charts (MSCs) from the MaSE models is as follows: Each role sequence diagram is searched for the roles which are listed in the same agent class shown in the agent class diagram. Following this all of the roles in each role sequence diagram are categorized based on their agent. Thus each category corresponds to an agent class of the agent class diagram and the messages which it exchanges with other categories are recognizable. From these two models an MSC can be generated which would display the recognized messages between each two categories.

2.2 Behavior Modeling and Detection of Emergent Behavior

Upon extracting Scenarios in the form of MSCs from MaSE models, they are used to model the behavior of the system. For an agent *i* of in an MSC m, a state machine [6] can be constructed. The behavior of agent *i* is modeled by the parallel execution of all resulted constructed state machines for *i*. 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 resulted state machine for agent *i* obtained through behavioural modelling [6]. State value calculations are conducted based on the principle of semantic causality [6]. Semantic causality approximately means that (a) a receive event cannot happen without having its corresponding send event happened before; and (b) an event cannot happen until all the causal predecessors of it have been accomplished [6].

3. DESIGN VERIFICATION TOOL

The devised methodologies devised in this research have been automated in a software tool. In the development of this software, there were two major concerns: scalability and usability. As there is still a vast amount of research remaining in this area, 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. 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 1) which closely represents the logical follow of the developed methodology.

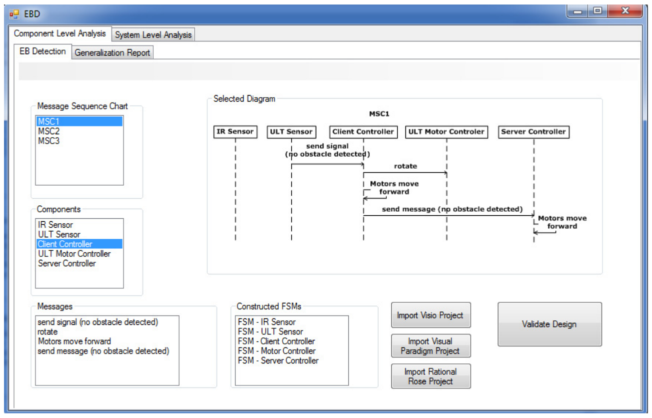

Figure 1. GUI for the input diagrams

4. CONCLUTIONS

Many failures in distributed systems are caused by subtle design faults introduced in the requirement elicitation and design phases of the software development life cycle. Detection of failures and removal of faults during field use of a system is about 20 times more expensive than detection and removal in the requirement and design phase [7]. The main objective of this research is to recognize potential design flaws that might lead to run time problems in distributed systems by analyzing the system requirements. However manual review is not always a feasible method to detect all the design faults due to the scale and complexity of the system. In this research we have devised techniques and developed a software tool to automate the requirements and design review of MAS and detect a subset of unwanted run time behaviors. For future work extending the proposed methodology to a comprehensive framework for model based analysis and testing of distributed software systems is indented. 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. As the developed tool closely follows the principle of encapsulations and modularization, it is indented to add the future results of this research to this tool upon completion.

5. REFERENCES

[1] J.M. Carroll, "Scenario-based design: envisioning work and technology in system development," Wiley, 1995.

[2] M. Cohn, "User Stories Applied," Addison Wesley, 2004.

[3] Unified Modeling Language. Available at http://www.omg.org/, 2002.

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

[5] S. A. DeLoach, "The MaSE Methodology," in Methodologies and Software Eng. for Agent System, pp. 107-147, Kluwer, 2004.

[6] M. Moshirpour, A. Mousavi, B. H. Far, "Detecting Emergent Behavior in Distributed Systems Using Scenario-based Specifications", International Journal of Software Engineering and Knowledge Engineering, Volume 13, No 6, Pages 729-746, 2012.

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