A UML-Based Conversion Tool for Monitoring and Testing Multi-Agent Systems

Nariman Mani *Vahid Garousi* *Behrouz H. Far*

Department of Electrical and Computer Engineering
Schulich School of Engineering, University of Calgary, Canada
{nmani, vgarousi, far}@ucalgary.ca

Abstract

The increasing demand for Multi-Agent Systems (MAS) in the software industry has led to development of several Agent Oriented Software Engineering (AOSE) methodologies. The autonomous agents' interaction in a dynamic software environment can potentially lead to runtime behavioral failures such as deadlock. Therefore, the MAS environment should be tested and monitored against the unwanted emergent behaviors. The AOSE methodologies usually do not cover monitoring and testing. On the other hand model-based software development practices such as the Unified Modeling Languages (UML) are commonly used in practice and are equipped with a rich set of model based testing and monitoring tools. In this paper, we propose a conversion tool to help MAS engineers use UML-based monitoring and testing tools to test and monitor MAS design and analysis artifacts created by Multi-agent Software Engineering (MaSE) as one of the most powerful and famous AOSE methodologies.

Index Terms— Multi-agent system, UML, Software testing, Deadlock detection.

1. Introduction

As a result of the growing demand in Multi-Agent Systems (MAS), many Agent Oriented Software Engineering (AOSE) methodologies have been evolved to assist the development of agent-based applications. Agent-based applications consist of the autonomous and intelligent software (agents) that can communicate and exchange information to solve challenging problems collaboratively[1]. An autonomous agent is a computational entity that can perceive, reason, act, and communicate [2]. A MAS consists of autonomous agents that try to achieve their goals by interacting with each other by means of high level protocols and languages [1]. Since the agents' interactions in MAS environment can potentially lead to behavioral faults, the MAS environment should be tested and monitored against the unwanted emergent behaviors. Also as model-based software development practices such as the Unified Modeling Languages (UML*)* [3] are gaining more popularity, more and more model based (UML-based) testing and monitoring tools are developed. UML is a language for specifying, constructing, visualizing, and documenting artifacts of software object-oriented systems [3]. There are several advantages to be gained from using the UML. Firstly, the UML provides high level information that illustrates the internal behavior of the system, which can be used efficiently and effectively in testing. Secondly, the UML has emerged as the de-facto industry standard for software modeling. Thirdly, the UML includes a set of models that can provide different levels of capacity and accuracy for modeling objects, and thus can be used to satisfy various needs in the real word industry [3].

The agent paradigm introduces a number of new abstractions and design/development concepts by means of AOSE methodologies to software development in comparison with regular model-based approaches such as UML. This makes the deployment of UML-based testing tools for verifying the internal behavior of MAS harder and sometimes impossible. Thus, conversion models that fill the gap between the AOSE design/ analysis concepts and the UML-based testing and monitoring tools can be very useful. These conversion models can help MAS engineers to deploy the UML-based testing and monitoring tools to test and verify the internal behavior of their developed MAS before bringing them to the main stream of commercial software.

In this paper we focus on proposing a conversion model and a prototype tool for adopting the MAS design and analysis models into standard UML models. We develop our proposed techniques based on the Multi-agent Software Engineering (MaSE) methodology design and analysis models [4] . MaSE is one of the AOSE methodologies which provides a detailed approach to the analysis and design of MAS. We show that the output of our proposed conversion model in this paper can be used in other applications, such as our previously published monitoring [5] and testing [6] multi-agent systems techniques for deadlock detection (Section 6).

The remainder of this paper is structured as follows. The related work and background are described in Section 2. A typical metamodel for MAS is introduced in Section 3. Constructing MAS behavioral model based on the MaSE analysis and design artifacts is discussed in Section 4. Constructing agent behavioral model based on the MaSE task diagram is described in Section 5. We discuss about conversion model application in monitoring and testing MAS in Section 6. Also, we provide a quick snapshot of the developed tool for conversion model in Section 7. Finally conclusions and future work are given in Section 8. Hypothetical and concrete examples are provided and used to explain the methodology in the subsequent sections.

2. Background and related work

In this paper we build our proposed methodology based on the MaSE methodology analysis and design artifacts as one of the AOSE methodologies. In this section, we first provide an overview of the MaSE methodology and its recent applications in research and industry (Section 2.1 and Section 2.2). Then we provide the results of an evaluation on MaSE methodology and its comparison with other AOSE methodologies in Section 2.3. Then we discuss other related works on MAS verification and monitoring and the methodology which they use in Section 2.4.

2.1. Agent Based Development Methodology: MaSE

MaSE uses several models and diagrams driven from the standard Unified Modeling Language (UML) to describe the architecture-independent structure of agents and their interactions [4]. The main focus in MaSE is to guide a MAS engineer from an initial set of requirements through the analysis, design and implementation of a working MAS. In MaSE a MAS is viewed as a high level abstraction of object oriented design of software where the agents are specialized objects that cooperate with each other via conversation and act proactively to accomplish individual and system-wide goals instead of calling methods and procedures. In other words, MaSE builds upon logical object oriented techniques and deploys them in specifications and design of MAS. There are two major phases in MaSE: analysis and design (Table 1).

Table 1- MaSE methodology phases and steps [4]

MaSE Phases and Steps	*Associated Models*
1. Analysis Phase	
a. Capturing Goals	Goal Hierarchy
b. Applying Use Cases	Use Cases, Sequence Diagrams
c. Refining Roles	Concurrent task, Role Diagram
2. Design Phase	
a. Creating Agent Classes	Agent Class Diagrams
b. Constructing Conversations	Conversation Diagrams
c. Assembling Agent Classes	Agent Architecture Diagrams
d. System Design	Deployment Diagrams

In Analysis phase [4], there are three steps which are capturing goals, applying use cases and refining goals (Table 1) . The MaSE Analysis phase produces a set of roles and tasks which describes how a system satisfies its overall goals. Goals are driven from the detailed requirements and should be achieved by defined roles. A role describes an entity that acts inside the system and is responsible for achieving, or helping to achieve specific system goals. In general, the main approach of MaSE analysis phase is to define system goals from a set of requirements and then define the roles necessary to meet those goals.

In the Design phase [4], there are four steps which are Creating Agent Classes, Constructing Conversations, Assembling Agent Classes and System Design (Table 1) . In the first step, Creating Agent Classes, the designer assigns roles to the specific agent types. During the second step, Constructing Conversation, the conversation between agent classes are defined while in the third step, Assembling Agent Classes, the internal architecture and reasoning processes of the agent classes are designed. Finally, in the last step, System Design, the designer defines the number and location of the agents in the deployed system.

2.2. MaSE Applications

MaSE has been successfully used in many agent-based research and industry applications. The Multi-Agent Distributed Goal Satisfaction project [7] is a collaborative effort between Air Force Institute of Technology (AFIT), the University of Connecticut, and Wright State University which uses MaSE to design the collaborative agent framework to integrate different constraint satisfaction and planning systems. MaSE has been also used successfully in agent-based heterogeneous database integration system [8] as well as a multi-agent approach to a biologically based computer virus immune system [9].

2.3. MaSE Comparison with other AOSE methodologies

There have been several methodologies for MAS analysis and design[2]. In [10], based on the set of criteria in terms of a hierarchy of dimensions and attributes that can be considered as empirical software metrics for evaluating AOSE methodologies, a set of 9 AOSE methodologies are evaluated. In [10], they were able to rank the evaluated methodologies according to the estimated mean effectiveness of the evaluation based on 6 dimensions which are Agency-related attributes, Modeling-related attributes, Communication-related attributes, Process-related attributes, Application-related attributes, and User perception attributes to support the decision of selecting the most appropriate methodology. In [10], MaSE is evaluated among 8 other AOSE methodologies and ranked 1st in 3 of proposed dimensions which are Modeling-related attributes, Application-related attributes, and User perception attributes. Finally, MaSE is ranked 1st in overall ranking of evaluated AOSE methodologies.

In this section, we only compare MaSE against the two other methodologies, Gaia [11] and Tropos [12]. As in MaSE, Gaia [11] uses roles as building blocks and both capture much of the same type of the information in design phase although in different types of models. The Gaia generates a high level design and assumes the details will be developed using other techniques whereas MaSE provides models and guidance on building the detailed design[2]. Tropos [12], takes a totally different approach in comparison with MaSE. Tropos focuses on early requirement which is not addressed in MaSE at all [2]. Although, the Tropos early requirements approach could be used in MaSE as the goal model in the design phase [2] [12].

2.4. MAS Verification and Monitoring

Existing works on MAS verification are categorized into axiomatic and model checking approaches [2]. In [13], axiomatic verification is applied to the Beliefs, Desires and Intentions (BDI) model of MAS using a concurrent temporal logic programming language. However, it was noticed that this kind of verification cannot be applied when the BDI principles are implemented with non-logic based languages [2]. Also in design by contract [14] pre- and post-conditions and invariants for the methods or procedures of the code are defined and verified in runtime. Violating any of them raises an exception. But as it is also claimed in [2] the problem is that this technique does not check program correctness, it just informs that a contract has been violated at runtime.

Model checking approaches seem to be more acceptable by industry, because of less complexity and better traceability as compared to axiomatic. Automatic verification of multi-agent conversations [15] and model checking MAS with MABLE programming language [16] are a few examples of model checking approaches that both use SPIN model checker [17], a verification system for detection of faults in the design models of software systems. But none of the mentioned approaches uses UML as their modeling techniques.

3. MAS Metamodel

Figure 1 shows a typical metamodel for the MAS structure. In this figure, each MAS can be presented by MAS behavioral model in terms of UML sequence diagrams which shows the conversations of several agents and the message exchanging among them (Section 4). The way of constructing such kind of behavioral model from MaSE design and analysis diagrams is proposed in Section 4.

On the other hand, each MAS consists of several agents whose roles are the building blocks used to define agent's classes and capture system goals during the design phase. Associated with each role are several tasks and each task can be presented by MaSE task diagram [4]. Each task diagram in MaSE can be converted to a UML state machine diagram which details how the goal is accomplished by a specific agent in a MAS and can be represented by a Control Flow Graph (CFG) [18] . More details on deriving CFG from MaSE task diagrams are provided in Section 5. A CFG is a static representation of a program that represents all alternatives of control flow. For example, a cycle in a CFG implies iteration. In a CFG, control flow paths (CFPs), show the different paths a program may follow during its execution.

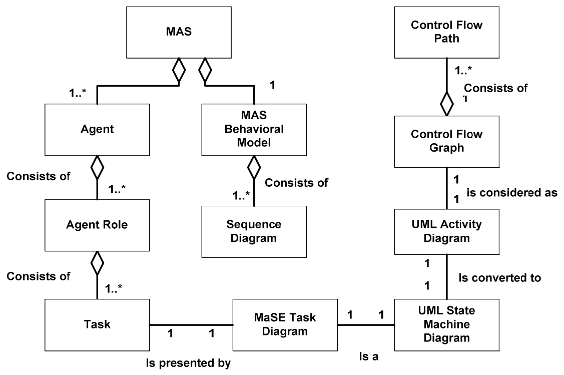


Figure 1- A metamodel for MAS

4. Constructing MAS behavioral model

The agents of a MAS communicate by exchanging messages. The sequence of messages is useful for understating the situation during faults detection conversations. A common type of interaction diagrams in UML is a sequence diagram in which each agent or role is represented by a lifeline in sequence diagram.

We use a method for transforming the conversations of agents from MaSE to UML sequence diagrams. These sequence diagrams are used in MAS monitoring method for deadlock detection in MAS under test [5]. The MAS sequence diagrams is not provided by MaSE per se and must be constructed using information provided by the MaSE artifacts such as role diagram and agent class diagrams [2].

The role sequence diagram in "Applying Use Cases" step in analysis phase of MaSE shows the conversations between roles assigned to each agent [4]. The agent class diagram is created in the "Constructing Agent Classes" step of MaSE. It represents the complete agent system organization consisting of agent classes and the high-level relationships among them. An agent class is a template for a type of agent with the system roles it plays. Multiple assignments of roles to an agent demonstrate the ability of agent to play assigned roles concurrently or sequentially. The agent class diagram in MaSE is similar to agent class diagram in object oriented design but the difference is that the agent classes are defined by roles, not by attributes and operations. Furthermore, relationships are conversations between agents [4]. Figure 2 and 3 show the hypothetical examples of MaSE role sequence and agent class diagram and the constructed behavioral model from them.

The approach for constructing sequence diagrams based on the two above mentioned MaSE diagrams is defined as follows. Each role sequence diagram is searched for the roles which are listed in the same agent class in the agent class diagram. Then, all of the roles in each role sequence diagram are categorized based on the agent which they belong to. Therefore, each category corresponds to an agent class in agent class diagram and the messages which it exchanges with other categories are recognizable. On the other hand, a new agent sequence diagram can be generated from agent class diagram which the lifelines are agents' types. The recognized messages between each two categories are entered into agent sequence diagram as a new conversation.

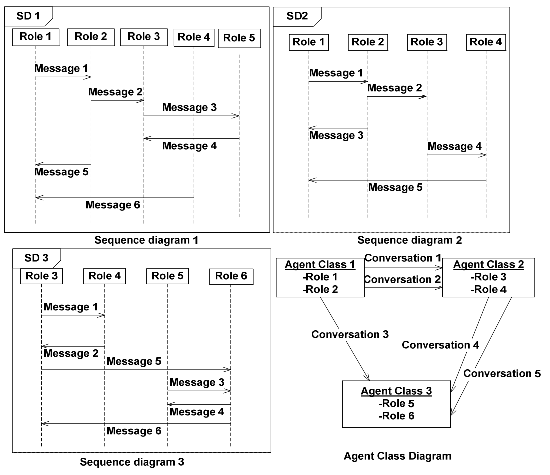


Figure 2- MaSE role sequence and agent class diagrams

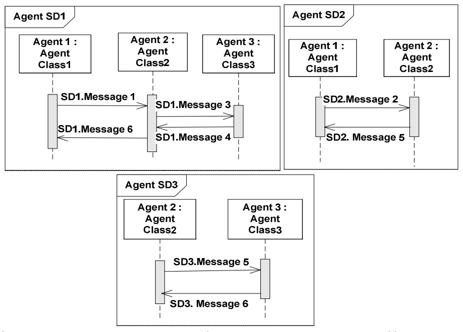


Figure 3- Constructed agent sequence diagrams

For example, in Figure 2, the role sequence diagram 1 is categorized into three different categories, the first one consists of Role 1 and Role 2 and the second one consists of Role 3 and Role 4 and the last one consists of Role 5. The first one corresponds to agent class 1, the second one corresponds into agent class 2, and the third one corresponds to agent class 3. The constructed agent sequence diagrams from role sequence diagram 1, 2 and 3 and agent class diagram in Figure 2 are shown in Figure 3.

Figure 4 and 5 represent a concrete example of constructing a MAS behavioral model from "role sequence diagram" and "agent class diagram" of "detection and notification of host's violations and logins ". In Figure 4, the "Agent Class diagram" with 5 agent classes and their roles and two "Role sequence diagrams" are provided. The constructed behavioral model from mentioned MaSE's diagrams is shown in Figure 5

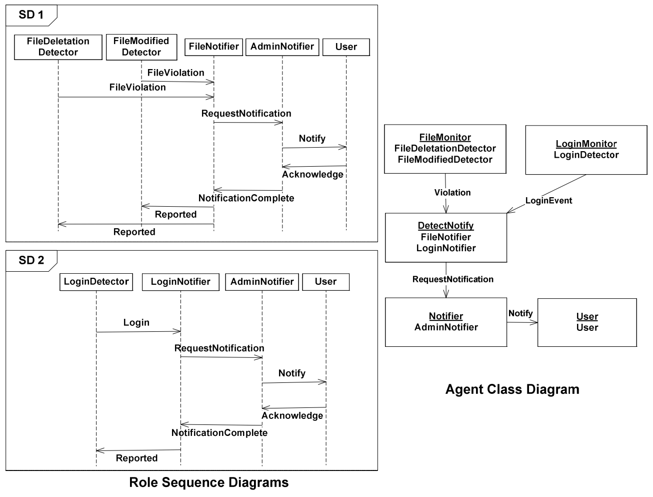


Figure 4- "Role sequence diagram" and "agent class diagram" for "detection and notification of host's violations and logins"

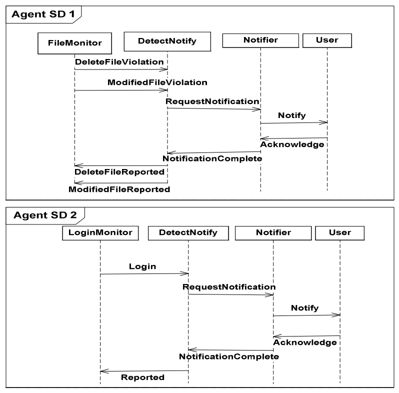


Figure 5- Constructed agent sequence diagrams for "detection and notification of host's violations and logins"

5. Constructing agent behavioral model

UML provides ways to model the behavior of an object oriented system using different types of diagrams such as state machine diagram. UML's state machine diagram is based on finite state machines (FSM) augmented with the concepts such as hierarchical and concurrent structure on states and the communications mechanism through events transitions. UML's state machine diagram is commonly used to describe the behavior of an object by specifying its response to the events triggered by the object itself or its external environment. State machine diagram has long been used as a basis for generating test data [19-21].

In MaSE [4], roles are the building blocks used to define agent's classes and capture system goals during the design phase. Every goal is associated with a role and every role is played by an agent class. Role definitions are captured in a role model diagram which includes information on communications between roles, the goals associated with each role, the set of tasks associated with each role, and interactions between role tasks. In MaSE, a task is a structured set of communications and activities, represented by a state machine diagram which consist of states and transitions[4]. States include the processing that goes on internal to the agent and transitions allow communication between agents or between tasks [4].

In this section, we first provide a comparison between MaSE task diagram and UML 2.0 state machine diagram and the differences in their transition protocol in Section 5.1. Then we discuss the procedure of deriving the activity diagram and its associated CFG from MaSE task diagram in Section 5.2.

5.1. MaSE task diagram vs. UML 2.0 state machine diagram

A transition in MaSE task diagram which as it is mentioned is state machine diagram uses the syntax of *trigger [guard] / transmission,* interpreted as if an event trigger is received and the condition guard holds, then the message transmission is sent [4]. In this transition notation all items are optional. However the UML 2.0 specification(Section 15.3 of [3]) proposed the syntax of *[precondition] event / [post condition]* for transition protocol in state machine diagram. The transition protocol specifies that when the associated (referred) operation is called because of an event in the origin state under the initial condition (pre condition or guard), then the destination state must be reached under the final condition (post condition) [3]. According to the above mentioned interpretation of the transition protocol in both MaSE task diagram and UML 2.0 state machine the *trigger* in MaSE task diagram can be considered as *event* in UML 2.0 state machine. Also, the *[Guard]* in MaSE task diagram can be considered as the *[precondition]* in UML 2.0 state machine diagram. Also, since the UML specification does not prescribe a syntax format for *[post condition]* (Section 15.3 of [3]), the *transmission* can be interpreted as the *[post condition]* in which the destination state must be reached under that. According to the mentioned mapping for protocol transition, MaSE task diagram can be converted to the UML 2.0 state machine diagram. Using the state machine diagram, the CFG and its associated CFPs can be identified [19, 20].

5.2. Deriving CFG from MaSE Task Diagram

As it is mentioned in Section 3, Control Flow Graph (CFG) [18] represents all alternatives of control flow in a program. The concept of CFG has been extensively deployed in the software engineering and particularly in the software testing community (e.g. [22, 23]).

UML has adopted a Petri-net like semantics for control and object flow modeling referred to as Activity Diagrams. Activity diagrams have been in UML since its early 1.x versions and they are used to describe both sequential and concurrent control flow and data flow [18]. As it is claimed by UML 2.0 (Section 12.1 of [3]), the UML activity diagrams are commonly called control flow and object flow models.

Figure 6 shows a task diagram for the Assign to Reviewers task. In MaSE Tasks diagram, states may contain activities that represent internal reasoning, reading a percept from sensors, or performing actions via actuators [4]. Multiple activities can be in a single state and are performed in an uninterruptable sequence [4]. Once in a state, the task remains there until the activity sequence is completed [4].

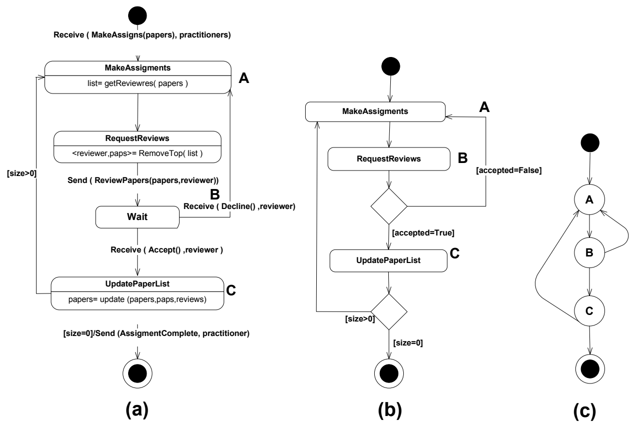

Receive (MakeAssigns(papers), practitioners)
MakeAssigments
list= getReviewres(papers)
A
RequestReviews
<reviewer,paps>= RemoveTop(list)
[size>0]
Send (ReviewPapers(papers,reviewer))
B
Receive (Decline() ,reviewer)
Wait
Receive (Accept() ,reviewer)
UpdatePaperList
papers= update (papers,paps,reviews)
C
[size=0]/Send (AssigmentComplete, practitioner)
(a)
MakeAssigments
A
RequestReviews
B
[accepted=False]
[accepted=True]
UpdatePaperList
C
[size>0]
[size=0]
(b)
A
B
C
(c)

Figure 6- (a) MaSE task diagram for Assign to Reviewers, (b) the corresponding activity diagram and (c) the associated CFG

So, the activities within tasks diagrams, their sequences, and their execution conditions can be driven from the states and the corresponding activity diagram for a MaSE task diagram can be created. Since the activity modeling is commonly called the control flow and object flow models [3] the corresponding CFG is also created by constructing the activity diagram. Also, since the protocol transition in MaSE task diagram uses the syntax of *trigger [guard] / transmission* and the *trigger* and *transmission* are limited to send and receive messages [4], trigger should be considered as the last activity of the source state and *transmission* should be considered as the first activity of destination state.

In this way, the *trigger* message is considered as the activity that after completing its execution the control flow will be transferred to the first activity of destination state (*transmission)*. Figure 6 shows (a) the Assign to Reviewers task for the Assigner role and (b) its corresponding activity diagram and (c) the associated Control Flow Graph (CFG).

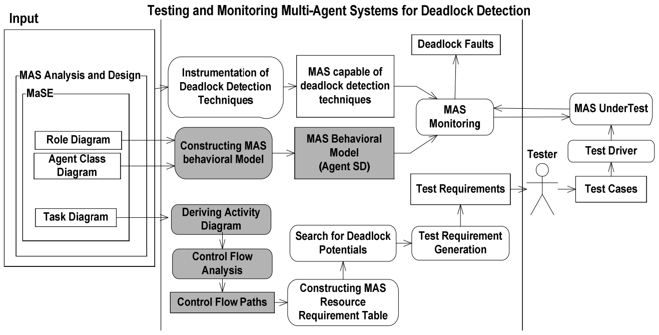
The task is initiated upon receipt of a make Assigns message from a Practitioner agent, which includes a list of papers to be assigned. After the message is received, the task goes to the Make Assignments state where it computes a list of reviewers for the papers. Once this list is defined, the task transitions to the “RequestReviews” state where the top reviewer/papers tuple is taken off the list. A “ReviewPapers” message is then sent to the reviewer effectively requesting that the agent provide a review for the associated papers, which is denoted by the “paps” parameter. The task remains in the Wait state until a reply from the reviewer is received. If the reviewer declines (via a decline message), the task returns to the “MakeAssignment” state where it computes a new list of reviewers for the remaining papers. If the reviewer accepts the request via an accept message, the task transitions to the “UpdatePaperList” state where the list of papers is updated by adding the name of the reviewer to the papers that they will be reviewing. If the list is not empty, the task returns to the “RequestReviews” state to make a request of the next reviewer on the list. If the size of the reviewers list is empty, the task ends by sending an “AssignmentComplete” message to the Practitioner agent.

In Figure 6 case, the *Wait* state is omitted in the corresponding activity diagram since it only can be considered as a state in state machine diagram and there is no activity assigned to it in MaSE task diagram.

6. Conversion model application in monitoring and testing MAS for deadlock detection

Figure 7 shows the role and application of the proposed conversion model in our monitoring [5] and testing [6] MAS for deadlock detection methodology. The grey boxes demonstrate the activities which are involved in conversion task.

The artifacts used are the models prepared during the analysis and design stages of a MAS using the MaSE methodology [4] . As it can be seen in Figure 7, using the procedure explained in [6], resource requirement table is constructed based on Control CFPs extracted from the MaSE task diagrams. The resource requirement table is used for searching for potential deadlocks [6]. Test requirements for testing MAS are prepared based on the potential deadlocks. The test requirements are used to generate the test cases. For deadlock detection on MAS under test we deploy our MAS monitoring methodology in [5]. Using the procedure explained in Section 4, a MAS behavioral model, consists of UML sequence diagrams, is constructed using MaSE analysis and design artifacts. Two deadlock detection techniques, introduced in [5], are instrumented into the MAS under test’s source code. Test driver executes the test cases on MAS under test and runtime deadlocks are detected using the MAS behavioral model [5] [6].

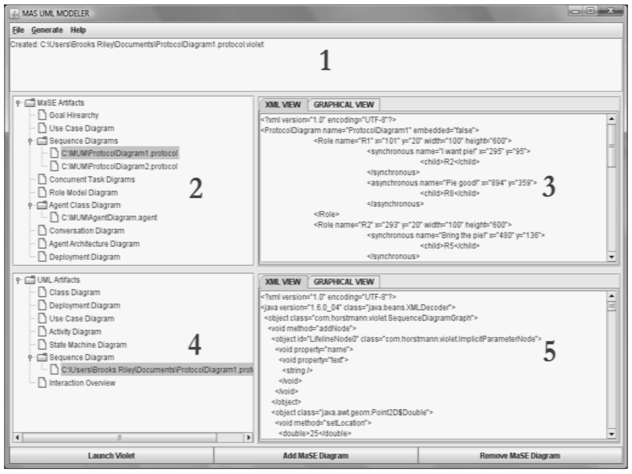


Input
Testing and Monitoring Multi-Agent Systems for Deadlock Detection
MAS Analysis and Design
MaSE
Role Diagram
Agent Class Diagram
Task Diagram
Instrumentation of Deadlock Detection Techniques
Constructing MAS behavioral Model
Deriving Activity Diagram
Control Flow Analysis
Control Flow Paths
MAS capable of deadlock detection techniques
MAS Behavioral Model (Agent SD)
Search for Deadlock Potentials
Constructing MAS Resource Requirement Table
Deadlock Faults
MAS Monitoring
Test Requirements
Test Requirement Generation
Tester
MAS UnderTest
Test Driver
Test Cases

Figure 7- Conversion model application in Testing and Monitoring MAS for deadlock detection

7. The developed tool for conversion model

Figure 8 demonstrates a snapshot of the developed tool for conversion model. Window 1, the Console, displays events that occur during tool operation such as diagram creation notifications, Error Messages, and Exceptions. Window 2, the MaSE Artifact Tree, shows all of the MaSE artifacts that are in the current conversion project (Inputs). This view can be modified by the “Add MaSE Diagrams” and the “Remove MaSE Diagram” at the bottom of the tool window. Window 3, the MaSE Artifact Viewer, shows the XML (XML is chosen as the input and output format for this tool) version of each selected MaSE artifacts in MaSE artifact tree.



Created: C:\Users\Brooks Riley\Documents\ProtocolDiagram1.protocol.violet

MaSE Artifacts
- Goal Hirearchy
- Use Case Diagram
- Sequence Diagrams
  - C:\MUM\ProtocolDiagram1.protocol
  - C:\MUM\ProtocolDiagram2.protocol
- Concurrent Task Digrams
- Role Model Diagram
- Agent Class Diagram
  - C:\MUM\AgentDiagram.agent
- Conversation Diagram
- Agent Architecture Diagram
- Deployment Diagram

```
<?xml version="1.0" encoding="UTF-8"?>
<ProtocolDiagram name="ProtocolDiagram1" embedded="false">
            <Role name="R1" x="101" y="20" width="100" height="600">
                        <synchronous name="I want pie!" x="295" y="95">
                                    <child>R2</child>
                        </synchronous>
                        <asynchronous name="Pie good!" x="894" y="359">
                                    <child>R8</child>
                        </asynchronous>
            </Role>
            <Role name="R2" x="293" y="20" width="100" height="600">
                        <synchronous name="Bring the pie!" x="480" y="136">
                                    <child>R5</child>
                        </synchronous>
```

UML Artifacts
- Class Diagram
- Deployment Diagram
- Use Case Diagram
- Activity Diagram
- State Machine Diagram
- Sequence Diagram
  - C:\Users\Brooks Riley\Documents\ProtocolDiagram1.prot
- Interaction Overview

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.6.0_04" class="java.beans.XMLDecoder">
 <object class="com.horstmann.violet.SequenceDiagramGraph">
  <void method="addNode">
   <object id="LifelineNode0" class="com.horstmann.violet.ImplicitParameterNode">
    <void property="name">
     <void property="text">
      <string/>
     </void>
    </void>
   </object>
   <object class="java.awt.geom.Point2D$Double">
    <void method="setLocation">
     <double>25</double>
```

Figure 8- The developed tool for conversion model

Window 4, the UML Artifact Tree, shows the UML diagram artifacts generated as the output of conversion model and finally window 5, The UML Artifact Viewer, demonstrates the XML version of the selected UML artifact in UML Artifact Tree.

8. Conclusion and future work

In this paper, we proposed a conversion model for adopting the MAS design and analysis models into standard UML models. This conversion tool helps MAS engineers use UML-based monitoring and testing tools to test and monitor MAS design and analysis artifacts created by Multi-agent Software Engineering (MaSE) as one of the most powerful and famous AOSE methodologies. We provided the results of an evaluation on MaSE methodology in Section 2.3 which shows that MaSE is ranked 1st among 8 other AOSE methodologies. We proposed a typical metamodel for MAS in section 3. In this metamodel we suggested a behavioral model for entire MAS and a behavioral model for the agents inside the MAS. An approach for constructing MAS behavioral model from MaSE role sequence diagrams and agent class diagram is presented in Section 4. Also, using the procedure explained in Section 5.2, we proposed agent behavioral model by extracting the Control Flow graph (CFG) and its associated Control Flow Paths (CFP) from the MaSE task diagrams in analysis phase of MaSE. In Section 6, we discussed the application of our constructed behavioral model of MAS and the behavioral model of agent in our previous monitoring [5] and testing [6] multi-agent systems for deadlock detection techniques. As the future work, we plan to apply our conversion model to a few more MAS case studies to evaluate its effectiveness and efficiency in monitoring and testing MAS techniques.

9. Acknowledgement

Nariman Mani, Vahid Garousi and Behrouz H. Far were supported by a discovery grant from NSERC. Vahid Garousi was further supported by an Alberta Ingenuity New Faculty Award. Haysam Alsayed, Lucian Aron, Tatyana Rabinovitch, and Brooks Riley helped the MAS-UML tool development. The authors acknowledge their effort and thank them collectively.

10. References

[1] M. R. Genesereth and P. K. Ketchpel, "Software agents " *Commun. ACM* vol. 37 pp. 48-53, 1994.

[2] F. Bergenti, M.P.Gleizes, and F. Zambonelli, *Methodologies and Software Engineering for Agent System* vol. 11. New York: Kluwer Academic Publishers, 2004.

[3] Object Management Group (OMG), "UML 2.1.1 Superstructure Specification," 2007.

[4] S. A. DeLoach, "The MaSE Methodology," in *Methodologies and Software Eng. for Agent System*. vol. 11, F. Bergenti, M.P.Gleizes, and F. Zambonelli, Eds. New York: Kluwer Academic Publishers, 2004, pp. 107-147.

[5] N. Mani, V. Garousi, and B. Far, "Monitoring Multi-agent Systems for Deadlock Detection Based on UML Models," in *21st IEEE Canadian Conference on Electrical and Computer Engineering (CCECE08) - Computer Systems and Applications* Niagara Falls , Canada, 2008.

[6] N. Mani, V. Garousi, and B. Far, "Testing Multi-Agent Systems for Deadlock Detection Based on UML Models," in *The 14th International Conference on Distributed Multimedia Systems (DMS08)* Boston, USA, 2008.

[7] G. M. Saba and E. Santos, "The Multi-Agent Distributed Goal Satisfaction System," in *ICSC Symposium on Multi-Agents and Mobile Agents in Virtual Organizations and E-Commerce*, 2000.

[8] J. T. McDonald, M. L. Talbert, and S. A. DeLoach, "Heterogeneous Database Integration Using Agent Oriented Information Systems," in *Proceedings of the International Conference on Artificial Intelligence*, 2000.

[9] P. K. Harmer and G. B. Lamont, "An Agent Architecture for a Computer Virus Immune System," in *Genetic and Evolutionary Computation Conference*, 2000.

[10] A. Elamy and B. Far, "A Statistical Approach for Evaluating and Assembling Agent-Oriented Software Engineering Methodologies," in *Agent-Oriented Information Systems IV*. vol. 4898/2008 Berlin / Heidelberg: Springer, 2008, pp. 105-122.

[11] M. Wooldridge, N. R. Jennings, and D. Kinny, "The Gaia Methodology for Agent-Oriented Analysis and Design," *Autonomous Agents and Multi-Agent Systems,* vol. 3, pp. 285-312, 2000.

[12] F. Giunchiglia, J. Mylopoulos, and A. Perini, "The tropos software development methodology: processes, models and diagrams," in *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, Italy, 2002, pp. 35 - 36

[13] M. J. Wooldridge and P. Ciancarini, "Agent-Oriented Software Engineering: The State of the Art," in *Proc. of the Workshop on Agent-Oriented Soft. Eng.*, 2000, pp. 1-28.

[14] B. Meyer, "Applying Design by Contract," *IEEE Computer,* vol. 25, pp. 40–51, 1992.

[15] H. L. Timothy and S. A. DeLoach, "Automatic Verification of Multiagent Conversations," in *the Annual Midwest Artificial Intelligence and Cognitive Science* Fayetteville, 2000.

[16] M. J. Wooldridge, M. Fisher, M. Huget, and S. Parsons, "Model Checking Multi-Agent Systems with MABLE,"

in *Proc. of the Int. Joint Conf. on Autonomous Agents and Multiagent Systems*, 2002, pp. 952–959.

[17] G. J. Holzmann, "The Model Checker Spin," *IEEE Trans. on Soft. Eng.,* vol. 23, pp. 279–295, 1997.

[18] V. Garousi, L. Briand, and Y. Labiche, "Control Flow Analysis of UML 2.0 Sequence Diagrams," in *Model Driven Architecture – Foundations and Applications*. vol. LNCS 3748/2005: Springer Berlin / Heidelberg, 2005, pp. 160-174.

[19] T. S. Chow, "Testing Software Design Modeled by Finite-State Machines," *IEEE Trans. on Software Eng.,* vol. 4, pp. 178-187 1978.

[20] H. S. Hong, Y. G. Kim, S. D. Cha, D. H. Bae, and H. Ural, "A test sequence selection method for statecharts," *Software Testing, Verification and Reliability,* vol. 10, pp. 203 - 227, 2000.

[21] L. C. Briand, Y. Labiche, and Q. Lin, "Improving Statechart Testing Criteria Using Data Flow Information," in *Proc. of the 16th IEEE Int. Symposium on Software Reliability Engineering*, Washington, DC, USA, 2005, pp. 95 - 104

[22] A. T. Chusho, "Test data selection and quality estimation based on the concept of essential branches for path testing," *IEEE Transaction on Software Engineering,* vol. 13, pp. 509-517, 1987.

[23] A. Bertolino and M. Marre, "Automatic Generation of Path Covers Based on the Control Flow Analysis of Computer Programs," *IEEE Transaction on Software Engineering,* vol. 20, pp. 885-899, 1994.