MONITORING MULTI-AGENT SYSTEMS FOR DEADLOCK DETECTION BASED ON UML MODELS

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

There is an increasing demand for Multi-Agent Systems (MAS) in the software industry. In order to bring MAS to the main stream of commercial software development, the behavior of MAS must be monitored and verified against the risk of unwanted emergent behaviors including deadlocks. In this paper, we introduce a methodology for efficient monitoring of MAS to detect resource and communication deadlocks. In this methodology, we construct a behavioral model of a MAS under analysis and use it for deadlock detection. The behavioral models are in the form of UML 2.0 sequence diagrams which are built from the modeling artifacts of the Multi-agent Software Engineering (MaSE) methodology. To detect MAS deadlocks at runtime based on UML sequence diagrams, we adapt and refine existing resource and communication deadlock detection techniques to the context of MAS. A monitoring scenario example of our methodology is presented.

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

1. INTRODUCTION

In software industry there is an increasing demand for applications which can communicate and exchange information to solve problems collaboratively. This has led to the growth of distributed software architecture consisting of several interoperable software systems. One of the main difficulties of interoperable software systems is heterogeneity. Different programs are written in different languages and by different programmers and must operate in dynamic software environment. Agent based software engineering is one of the approaches devised to handle collaboration and interoperability. Organizations that have successfully implemented agent technologies include DaimlerChrysler, and IBM [1]. A MAS consists of autonomous software agents that try to achieve their goals by interacting with each other by means of high level protocols and languages [2] . However, the agent interaction can potentially lead to runtime behavioral faults including deadlock.

In order to bring MAS to the main stream of commercial software the internal behavior of the MAS must be monitored and verified to eliminate the risk of unwanted emergent behavior. Monitoring and verification usually consists of checking for communication faults such as deadlocks, infinite loops, livelocks and other communication pitfalls. As model-based software development practices are getting more popularity [3], more and more MAS are developed using model-based practices such as the Multi-agent Software Engineering (MaSE)[4]. Thus, model-based monitoring techniques in the context of MAS can be useful since they can use the existing models which have been built for analysis and design of MAS and can help MAS engineers to make sure deadlocks are detected as soon as they occur at runtime. As discussed in [5], there are only a few model based techniques to detect MAS deadlocks at runtime.

In this paper we focus on model based deadlock detection by checking MAS communication for existence of deadlock. The artifacts used are the models prepared during the analysis and design stages of a MAS using the MaSE methodology[4]. Figure 1 illustrates our approach. Using the procedure explained in Section 3, a MAS behavioral model, consists of UML sequence diagrams, is constructed based on "role sequence diagram" and "agent class diagram" built during the MaSE analysis and design stages. Two deadlock detection techniques (see Section 4) are then instrumented into the MAS source code. The MAS monitoring module runs the instrumented MAS system and uses MAS behavioral model at the runtime to detect deadlocks. The reported deadlock faults based on the runtime system feedbacks are provided by MAS monitoring module as the result which can be used for debugging and corrections purposes. The methodology can be used as a monitoring method for deadlock detection in both testing phase of MAS development and MAS deployment.

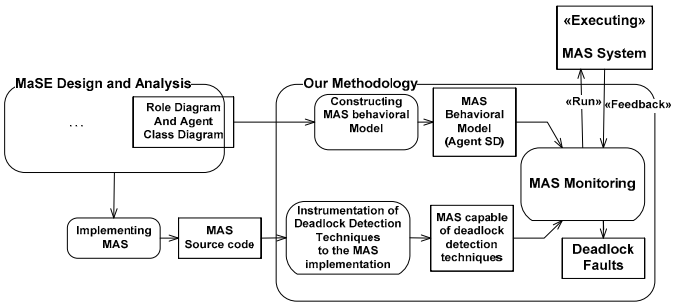

Figure 1- An overview of MAS deadlock monitoring methodology

The remainder of this paper is structured as follows. The related work and background are described in section 2. Constructing MAS behavioral model based on the MaSE is discussed in section 3, deadlock detection techniques in resource and communication deadlock models are introduced in section 4. Finally conclusions are given in section 5. An illustrated example of agent behavioral model in terms of sequence diagrams is constructed in section 3 and is used to explain the methodology in the subsequent sections.

978-1-4244-1643-1/08/$25.00 ©2008 IEEE

2. RELATED WORK AND BACKGROUND

2.1 MAS Verification and Monitoring

Existing work on MAS verification is categorized into axiomatic and model checking approaches [5]. In [6], 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 [5]. Also in design by contract [7] 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.

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 [8] and model checking MAS with MABLE programming language [9] are a few examples of model checking approaches that both use SPIN model checker [10], a verification system for detection of faults in the design models of software systems.

2.2 Deadlock Detection Techniques

Resource and communication deadlocks models are considered in message communication systems. Most deadlock models in distributed systems are resource models [11, 12]. In these models, the competition is on acquiring required resources and deadlock happens whenever an entity is waiting permanently for a resource which is held by another. As indicated in [12], the communication deadlock model is general and can be applied to any message communication system. The communication model is an abstract description of a network of entities which communicate via messages. A deadlock detection mechanism based on the communication deadlock model is provided in [12]. In literature a deadlock situation is usually defined as "A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause" [13]. There are four conditions that are required for a deadlock to occur [13]. They are (1) "Mutual Exclusion" which means each resource can only be assigned to exactly one process; (2) "Hold and Wait" in which processes can hold a resource and request more; (3) "No Preemption" which means resources cannot be forcibly removed from a process; and (4) "Circular Wait" which means there must be a circular chain of processes, each waiting for a resource held by the next member in the chain [13]. Similar to other types of the faults there are four techniques commonly used to deal with deadlock problem: ignorance, detection, prevention and avoidance [13]. The approach in this paper is a deadlock detection technique which uses the behavioral model of MAS to monitor running system to detect deadlocks.

2.3 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]. 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 instead of calling methods and procedures. There are two major phases in MaSE: analysis and design (Table 1). In analysis phase, there are three steps which are capturing goals, applying use cases and refining goals. In the design phase, there are four steps which are creating agent classes, constructing conversations, assembling agent classes and system design [4].

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

3. CONSTRUCTING MAS BEHAVIOURAL MODEL

Agents of a MAS communicate by exchanging messages. The sequence of messages is useful for understating the situation during faults detection conversation. A common type of interaction diagrams in UML is sequence diagram in which each agent or role is represented by a participating object (lifeline in sequence diagram). In this work the conversations of agents are transformed from MaSE role diagrams to UML sequence diagrams. We then monitor the given MAS based on its sequence diagrams and detect deadlocks.

In this section we devise a method to construct a sequence diagram which shows the conversations of several agents and the message exchanging among them. Unfortunately, such a diagram is not provided by MaSE and it should be constructed based on the information available in the other MaSE diagrams. The only sequence diagram provided by MaSE is role sequence diagram in "Applying Use Cases" step in analysis phase which shows the conversations between roles assigned to each agent [4].

Another diagram which we use to construct agents sequence diagram is a MaSE agent class diagram. The agent class diagram which is created in the "Constructing Agent Classes" step of MaSE 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 shows examples of MaSE role sequence and agent class diagram.

The approach for constructing agent sequence diagram based on the two above mentioned MaSE diagrams is defined as follow. 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. For example, in Figure 2, the role sequence diagram 1 is categorized into three different categories, one consists of Role 1 and Role 2 and the other one consists of Role 3 and Role 4 and finally one consists of Role 5. The first one corresponds to agent class 1 and the second one corresponds into agent class 2 in agent

class diagram and the third one corresponds to agent class 3. Constructed agent sequence diagrams from role sequence diagram 1, 2, 3 and agent class diagram in Figure 2 is shown in Figure 3.

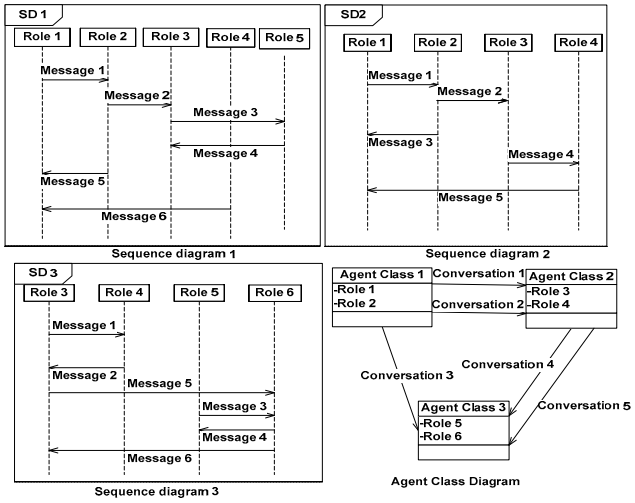

Figure 2- MaSE role sequence and agent class diagrams

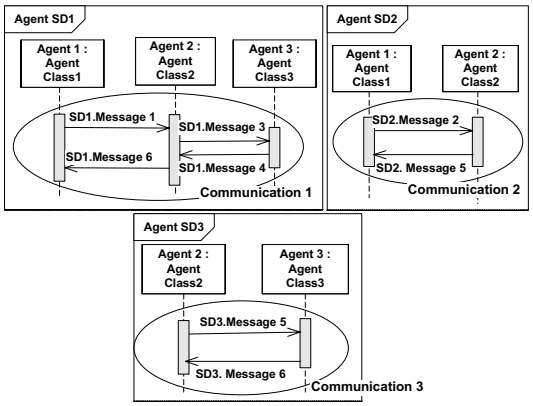

Figure 3- Constructed agent sequence diagrams

The definition of conversation in constructed sequence diagrams which illustrates the behavior of MAS should be clear. We define *Conversation* as a message exchange between two specific agents for a special purpose. Each *conversation* has two agents involved. In the constructed sequence diagrams a specific *conversation* for each agent is shown by execution occurrences specification or activity boxes (grey boxes on the lifeline).

There are several processes executing on the MAS which all are handled by agents. Several agents communicate with each other to handle tasks of a specific process. Conversation A is related to conversation B, if conversation A exchanges at least one message with conversation B during its execution. We call a set of related conversations a "communication" which handles a process or a part of it. Each communication may consist of several conversations. As an example in Figure 3, communication 1, 2 and 3 are identified.

4. DEADLOCK DETECTION IN MAS

The resource and communication deadlock models in MAS are discussed in this section and deadlock detection techniques are presented using the ideas from [12].

4.1 Deadlock Detection in Resource Deadlock Model

Resource model of MAS consists of agents, resources and controller agents. Controller agents are responsible for resource allocation. A controller agent is associated with a set of agents and a set of resources which it manages. Agents request for acquiring resources from their controller. Also, the controller can communicate with other controllers in case of requesting resources from other controllers. We assume that in the resource deadlock model each resource can only be assigned to exactly one agent and each agent can hold a resource and request more and finally resources cannot be forcibly removed from an agent. Several processes are active in a MAS and tasks associated with each process are performed by agents' cooperation. Each agent cannot perform the process task which is assigned to it unless it acquires all of the resources which it needs. We define a set of agents as *deadlocked* when no agent in the set can perform its task regarding a specific process because that task needs a resource which is acquired by another agent in the set. The controller agent can keep track of all resource allocations therefore behavioral faults are detectable by analyzing information collected by controller agent. However, it is possible for an agent to request for acquiring a resource from another controller. In such a case if the agent identifies itself in waiting state, it is not clear whether it can acquire the requested resource ever. The reason is that its request may be passed to other controllers and agents which can release that required resource for it. One possible solution is the MAS negotiation. But there are three main reasons for why agents in heterogeneous MAS are not completely interoperable. These reasons are called architectural elements[14] and are as follow: (1) inconsistent mental state structures; (2) different syntax and semantics of the agent communication languages; and (3) incompatible message transport mechanisms. For tackling these problems, a gateway agent can be proposed as a translator of the agent communications. A prototype of the gateway agent that can translate messages between Knowledge Query Manipulation Language (KQML) [15] and FIPA Agent Communication Language speaking agents (FIPA-ACL) [16] is developed in [14]. In this work, a communication protocol is defined for controller agents to communicate, acquire resources and handle behavioral faults such as deadlock.

Whenever agent A_a in controller C_i needs to acquire a resource R_i associated to another controller C_j, it sends its request to its controller C_i. C_i communicates with controller C_j regarding the requested resource R_i. If the required resource is available, C_j provides the resource for agent A_a in controller C_i .But if it is held by another agent A_b, C_i provides the identification of agent A_b to controller C_i . So , each controller agent has information about the internal resource allocation inside its set and the external resources that each agent in its set has already acquired or wants to acquire.

In resource model, agent A_1 is said to be dependent on another agent A_k if there exists a sequence of agents $A_1, A_2, \ldots, A_k$ where each agent in sequence is idle and each agent in sequence except the first one holds a resource for which the previous agent is waiting. Agent A_1 is internally dependent on A_k if both are in the same MAS, i.e. both associated to one controller agent. If A_1 is dependent on A_k, then A_1 has to remain in idle status as long as A_k is idle. A_1 is deadlocked if it is dependent on itself or on an agent which is dependent on itself. The deadlock can be extended to a cycle of idle agents, each dependent on the next in the cycle. Therefore, the resource model deadlock detection is to declare the existence of that cycle.

In order to determine for an idle agent A_a whether it is in deadlock state or not, a communication is initiated by its controller agent. In deadlock detection communication, controller agents send an *investigator* message to each other. An *investigator* message is of the form $Investigator(n,m,a,b,r,c)$ denoting that it is initiated by controller of agent A_a for process P_n and transaction T_m regarding agent A_b which requested to acquire resource R_r that it is currently held by A_c . $Investigator(n,m,a,b,r,c)$ is sent by controller C_i to controller C_j when one of the following conditions is true: (1) A_b associated to controller C_i is idle and A_b is waiting to acquire resource R_r associated to the controller C_j (externally) and R_r is currently held by A_c associated to the controller C_j , (2) A_b associated to controller C_i is idle and A_b is waiting to acquire resource R_r associated to the controller C_i (internally) and R_r is currently held by A_c associated to the controller C_j. $Investigator(n,m,a,b,r,c)$ is accepted by C_j if the A_c is idle and has not know that A_a is dependent on it (and know it knows) and A_c is waiting to acquire a resource (internally or externally) and that resource is currently held by another agent. In this condition C_j send an investigator message to the controller of A_c. It follows that if C_i receives $Investigator(n,m,a,b,r,a)$ from another controller for any possible b and r and if R_r is one of the resources which is held by A_a, a circular wait is detected and C_i declare A_a as deadlocked. Pseudo code for deadlock detection in resource model of MAS is shown in Figure 4. To reduce the number of deadlock detection computations which are initiated, an agent may initiate one only if it has been idle continuously for some time T, where we call T as a performance parameter.

As an example for deadlock detection in resource deadlock model, we consider the following scenario. Agent A_1 is holding resource R_1 associated to its controller C_1 and requested to acquire R_2 associated to controller C_2. A_2 in C_2 is holding R_2 and requested to acquire R_3 from C_3. A_3 in C_3 is holding R_3 and requested to acquire to acquire R_3 which is held by A_1 . The wait-for graph [17] of this example and the agent sequence diagram which shows the message communication between controllers for deadlock detection in the resource model is shown in figure 5. According to our assumptions for resource deadlock model, three of four deadlock conditions are true in this example which are (1) "Mutual Exclusion"; (2) "Hold and Wait"; (3) "No Preemption". Also as it can be seen in figure 5 the fourth condition circular wait can be detected after receiving $Investigator(n,m,\ A_1,A_3,R_3,A_1)$ by C_1 and identifying that R_3 requested by A_3 and is held by A_1. So all the four condition of deadlock is true and C_1 can declare A_1 as deadlocked.

4.2 Deadlock Detection in Communication Deadlock Model

In the communication deadlock model of MAS there is no controller agent or resources. Associated with each idle agent is a set of dependent agents which is called its *dependency set*. Each agent in that set can change its state about one particular task from idle to executing upon receiving a message from one of the members of its dependency set regarding that task.

We define a nonempty set of agents as deadlocked if all agents in that set are permanently idle regarding a special task. An agent is called permanently idle, if it never receive a message from its dependency set to change its state. In the more precise definition, a none empty set of agents S is called deadlock if and only if all agents in S are in idle state, the dependency set of every agent in *S* is a subset of *S* and there are no active conversation between agents in *S*. Based on this definition, all agents in the set can be called permanently idle, because the dependency set of each agent such as A_i is in *S*, so they are all in idle state, and also there is not any trigger message in transit between agents just because there is no active conversation.

A controller on initiation of a investigator for an idle agent A_a (after a defined time *T)*
If A_a is locally dependent on itself or on a set of idle agents all in same controller
Then *Declare* A_a *deadlocked*
Else
For all agents (in parallel) such as A_a which want to acquire resource *r* (internally or externally) which is held by another agent A_c in another controller,
Send $Investigator(n,m,a,a,r,c)$ *to controller of agent* A_c
End For
End If
A controller on receiving a $Investigator(n,m,a,b,r,c)$
If A_c is idle, and it has not replied to all requests of A_b recently (after a defined time *T*), and A_c has not known that A_a is dependent on it.
Then
Declare *that* A_a *is dependent on* A_c
If $a = c$ and R_r is one of the resources which are held by A_a
Then *declare that* A_a *is deadlocked*
Else
For all agents (in parallel) such as A_c which want to acquire resource R_r (internally or externally) which is held by another agent A_e in another controller,
Send $Investigator(n,m,a,c,r,e)$ *to controller of agent* A_e
End For
End If
End If

Figure 4- Pseudo code for deadlock detection in resource deadlock model

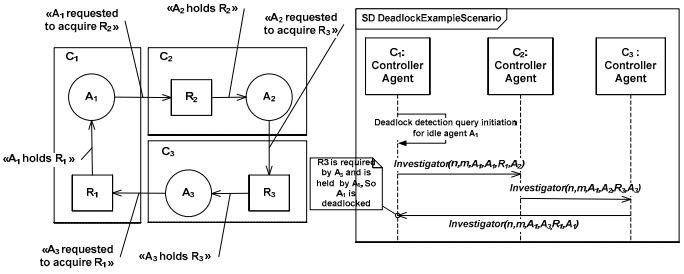

Figure 5 - deadlock detection example in resource deadlock model

An idle agent can determine if it is deadlocked by initiating a deadlock detection conversation with its dependency set when it enters to idle state. Each agent can initiate many deadlock detection communication which are numbered by m. An agent A_i is deadlock if and only if it initiates a query conversation to all the agents in its dependency set and receives reply to every query that it sent. The purpose of initiating this query is to find out if the agent A_i belongs to a deadlock set *S* with the mentioned conditions above. On receiving a query by an idle agent in dependency set, it should forward the query to its dependency set if it has not done already. For example if there is sequence of idle agents such as $A_1, A_2,\ldots,A_k$ that each one is in the dependency set of the previous one, a query initiated by A_1 will be forwarded to A_k.

Each query has the form $DeadlockDetectionQuery(n,m,i,a,b)$ denoting this message belongs to mth deadlock detection communication initiated for nth communication in MAS by agent A_i which is sent from agent A_a to agent A_b .Each agent A_k keeps the latest deadlock detection communication which it has been participated in it by $LatestDeadlockDetectComm(i)$ denoting the latest deadlock detection communication number that A_k was

participated in it and initiated by A_i. The state of each agent (idle/executing) also is stored by $State(n, m, i)$ denoting the state of agent A_k for mth deadlock detection communication initiated by A_i for nth communication in MAS. Also the number of replies which should be received by an agent for mth deadlock detection communication initiated by A_i for nth communication in MAS is stored in $NumOfReplies(i)$. Algorithm for communication model deadlock in of MAS is shown by pseudo code in Figure 6.

An idle agent A_i to initiate a deadlock detection
query $DeadlockDetectionQuery(n, m, i, a, b)$: (after a defined time T)
Begin
$LatestDeadlockDetectComm(i) = LatestDeadlockDetectComm(i) + 1$;
$State(n, m, i)$ = waiting;
Send $DeadlockDetectionQuery(n, LatestDeadlockDetectComm(i), i, i, b)$ to all agents A_b in A_i's dependent set (see section 4.3) S in parallel ;
$NumOfReplies(i)$= Number of agents in A_i's dependent set S
End
An idle agent A_b , upon receiving $DeadlockDetectionQuery(n, m, i, a, b)$:
if m > $LatestDeadlockDetectComm(i)$
Then begin
$LatestDeadlockDetectComm(i) = m$;
$State(n, m, i)$ = waiting;
for all agents A_r in $A_b's$ dependent set S (see section 4.3) in parallel
Send $DeadlockDetectionQuery(n, m, i, b, r)$;
$NumOfReplies(i)$= Number of agents in A_i's dependent set S
End for
Else if $State(n, m, i)$ = *waiting* and m = $LatestDeadlockDetectComm(i)$
then send $Reply(n, m, i, b, a)$ to A_a
End if
Upon receiving $Reply(n, m, i, c, b)$ by A_b.
if m = $LatestDeadlockDetectComm(i)$ and $State(n, m, i)$= *waiting*
then begin
$NumOfReplies(i)$:= $NumOfReplies(i)$ - 1;
if $NumOfReplies(i)$=0
then if i = b
Declare A_b deadlocked
End if
else send $Reply(n, m, i, a, r)$ to A_r
where A_r is the agent which caused $LatestDeadlockDetectComm(i)$ be set to its current value
End if
End if

Figure 6- Pseudo code for deadlock detection in communication deadlock model

We present the following scenario as an example for deadlock detection in the communication deadlock model of a hypothetical MAS with the sequence diagrams shown in Figure 3. In this scenario, agent A_1 is executing and has not received any message in one of its communications (communication 1 in this case) for a defined time T from its dependency set $\{A_2, A_3\}$(Section 4.3). This is since A_2 and A_3 are both in waiting state and A_1 is not aware of it. After the defined time T , A_1 identifies itself as an idle agent and initiates a deadlock detection conversation with each agent in its dependency set. An agent A_1 declare itself as deadlocked if and only if it initiates a query conversation to all the agents in its dependency set and receives reply to every query that it had sent. The complete deadlock detection scenario for the mentioned scenario is shown as a sequence diagram in figure 7. A_1 initiates two query conversations with its dependency set which are A_2 and A_3 . A_2 and A_3 propagate the queries to their own dependency set which are$\{ A_1, A_3 \}$ for A_2 and $\{ A_1 , A_2\}$ for A_3 . Respectively A_2 and A_3 receive replies from their own dependency sets. Thus they both reply to A_1 which is the initiator of deadlock queries in this scenario. So, A_1 receives reply for all quires which it had initiated to its dependency set. So, ir declares itself as deadlocked.

4.3 Dependency set identification

The deadlock detection techniques described in section 4 rely on identifying the dependency set for an agent. The agent A_i is member of dependency set of agent A_a if A_i is structurally or functionally dependent to A_a [18]. From UML 2.0 point of view, dependency is defined as "*a relationship that signifies that a single or a set of model elements requires other model elements for their specification or implementation. This means that the complete semantics of the depending elements is either semantically or structurally dependent on the definition of the supplier element(s)*"[3].

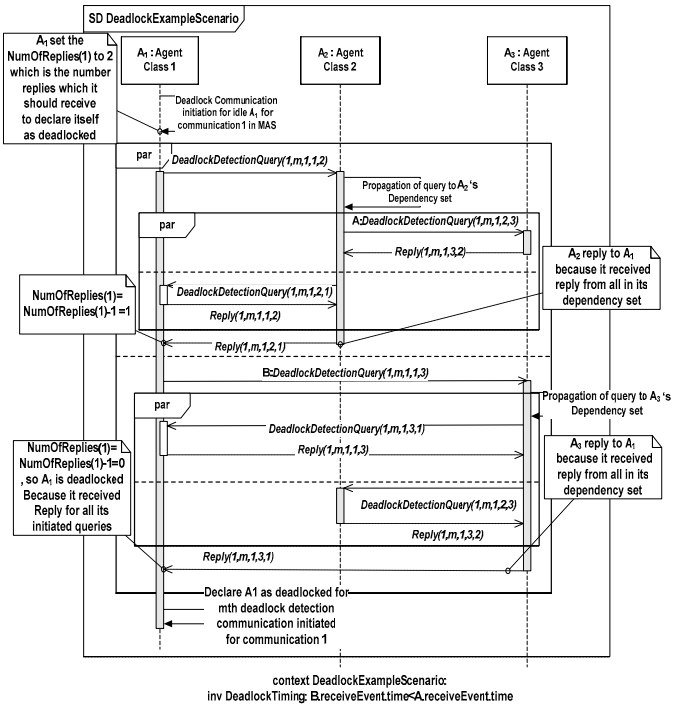

Figure 7 – A deadlock detection scenario in communication deadlock model using our technique

We can conclude that the dependency can be categorized into two groups which are structural and behavioral. In the structural dependency, the implementation changes in one entity can affect another entity. In behavioral dependency one entity requires a service from another entity in order to complete its assigned task [18]. The focus of this writing is on behavioral faults such as deadlock in agent communications and this point of view leads to the dependency of agents based on their behavior. Behavioral dependencies in the context of MAS can be of several types [18]:

- *A data dependency* exists whenever the available data for one agent is required in another one for completing of its task.
- *Time dependency* exists when the behavior of one agent in MAS has to precede or follow the behavior of another one to complete its task.
- *State dependency* specifies that the behavior of an agent will not happen unless the system or some part of the system, or also another agent is in a specified state.
- *Causal dependency* specifies that the behavior of an agent in MAS entails a specific behavior for another agent.

As mentioned earlier, the assumption is that the processes and associated tasks in heterogeneous MAS are handled by means of communications between agents. Whenever an agent identifies itself in waiting or idle state for a specific communication, it initiates a deadlock detection communication query or propagates a query to its dependency set. As it is clear all types of behavioral dependencies which are mentioned above can exist between agents which are somehow involved in a specific conversation. So the basic and immediate approach for an agent in MAS is to identify all of the agents which are involved in a specific communication as its dependency set whenever it declares itself waiting or idle for that communication. In Figure 3, *Agent 1* initiate a deadlock query communication or propagate a query to *Agents 2* and *Agent 3* as it dependency set.

4.4 Comparison between resource and communication models

The main difference between resource and communication models is that the identifications of agents that communicate with a specific agent are known in the communication model. On the other hand, an agent can identify the agents from which it has to receive message before it can continue its assigned task in the communication model. For example, if agent A expects to receive a message from agent B, the former recognize its status that it is waiting for agent B. So, the agent has most of the required information for deadlock detection approach if it finds itself in idle situation. But in the resource model the dependency of one agent action on actions of others is not directly noticeable. All is known is the dependency of one agent action for performing the assigned task to a specific resource, but agents which can release that required resource are not known. In such a case, the controller agent of each site (a set of agents) should keep track of the resource allocation on that site and collect information about the dependency of agent actions or tasks with respect to each other. The conclusion is that different deadlock detection approaches should be used for resource and communication models.

5. CONLUCSION AND FUTURE WORK

In this paper, we presented a methodology for runtime monitoring of the behavior of a Multi-Agent System (MAS) to detect deadlocks as one of its unwanted emergent behavior. Our methodology uses a behavioral model of the MAS, i.e. UML sequence diagrams constructed from MaSE analysis and design artifacts. An approach for constructing UML agent sequence diagrams from MaSE role diagrams is presented in section 3. We instrument the source code of the system with specific instructions in terms of deadlock detection techniques to enable runtime deadlock detection. To design our resource and communication deadlock detection techniques, we use and refine the approaches from the work in [12].

In order to efficiently detect deadlocks and only involve the needed agents in the deadlock detection mechanism, our technique relies on identifying the dependency set for each agent (Section 4.3). To account for the above efficiency criteria, the focus of this paper is on indentifying the dependency set based on the involved agents in a communication regardless of the dependency type. As the future work, the tradeoffs between effectiveness and efficiency of our deadlock detection techniques will be analyzed with regards to different types of dependencies mentioned in section 4.3. Also, our methodology will be applied to a few more MAS case studies to evaluate its effectiveness and efficiency.

ACKNOWLDGEMENTS

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.

REFERENCES

[1] H. V. D. Parunak, "A Practitioners' Review of Industrial Agent Applications," *Journal of Autonomous Agents and Multi-Agent Systems,* vol. 3, pp. 389-407, 2000.

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

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

[6] 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.

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

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

[9] 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.

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

[11] V. Gligor and S. Shattuck, "Deadlock detection in distributed systems," *IEEE Trans. Soft. Eng.,* pp. 435-440, 1980.

[12] K. M. Chandy, J. Misra, and L. M. Haas, "Distributed Deadlock Detection," *ACM Trans. on Computer Systems,* vol. 1, pp. 144-156, 1983.

[13] A. Tanenbaum, *Modern Operating Systems*. Englewood Cliffs: Prentice Hall Inc., 1992.

[14] H. Suguri, E. Kodama, M. Miyazaki, and I. Kaji, " Assuring interoperability between heterogeneous multi-agent systems with a gateway agent," in *Proc. of the 7th IEEE Int. Symp. on High Assurance Systems Eng.*, 2002, pp. 167-170.

[15] T. Finin, R. Fritzson, D. McKay, and R. McEntire, "KQML as an agent communication language," in *The Third Int. Conf. on Information and Knowledge Management*, Gaithersburg, Maryland, United States, 1994, pp. 456 - 463

[16] Foundation for Intelligent Physical Agents (FIPA), "FIPA ACL Message Structure Specification," 2002.

[17] D.A. Menasce and R.R. Muntz, "Locking and Deadlock Detection in Distributed Data Bases," *IEEE Transactions on Software Eng.,* vol. 5, pp. 195-202, 1979.

[18] V. Garousi, L. Briand, and Y. Labiche, "Analysis and Visualization of Behavioral Dependencies among Distributed Objects based on UML Models," in *Proc. of Int. Conf. on Model Driven Engineering Languages and Systems*, 2006, pp. 365-379.