2009 21st IEEE International Conference on Tools with Artificial Intelligence

Runtime Monitoring of Multi-Agent Manufacturing Systems for Deadlock Detection Based on 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 the Multi-Agent Systems (MAS) in the automation of manufacturing systems. However, similar to other distributed systems, autonomous agents' interaction in the Automated Manufacturing Systems (AMS) can potentially lead to runtime behavioral failures including deadlock. Deadlocks can cause major financial consequences by negatively affecting the production cost and time. Therefore, a multi-agent manufacturing system should be monitored against the unwanted emergent behaviors such as deadlocks. In this paper, we propose a monitoring technique for deadlock detection in multi-agent manufacturing system based on the MAS design models. In this technique, the MAS is instrumented with a dedicated communication protocol to use the potential deadlock information derived from the design models to propagate deadlock detection query messages. The technique is able to reduce the message communication overhead among the agents by limiting the number of agents that the deadlock detection query messages should be initiated to.

Keywords: Multi-Agent Systems (MAS), Automated Manufacturing Systems (AMS), Deadlock, Monitoring, Multi-Agent Software Engineering (MaSE).

1. Introduction

An Automated Manufacturing System (AMS) is an integrated system of equipment and processes controlled via computer applications or a network of them that is capable of producing a variety of products with flexibility and efficiency [1]. The agent based software engineering methodologies has been widely used in several AMS applications such as concurrent engineering, collaborative engineering design, manufacturing enterprise integration, supply chain management, manufacturing planning, scheduling and control, and material handling [2]. An agent is a computational entity that can perceive, reason, act, and communicate autonomously [2]. A manufacturing system automated by agent based technology is composed of several autonomous and intelligent agents that can communicate and exchange information to manage the product line processes and solve challenging problems collaboratively. A system composed of multiple interacting intelligent agents is called Multi-Agent System (MAS) [3]. Therefore, in this paper we call a manufacturing system automated by means of several interacting agents as multi-agent manufacturing system.

Similar to other distributed computing systems, multi-agent manufacturing systems are prone to the conflicts such as deadlock situations, wherein two or more competing agent actions are waiting for the other to finish, and thus neither ever does. In multi-agent manufacturing systems, the agents are responsible for managing the production jobs such as resource allocation to the production tasks. Considering machines and robots as the system resources in the multi-agent manufacturing systems, deadlock arises when resources are allocated to the production tasks in a way that makes task flow impossible. This can cause the major interruption to the manufacturing process by affecting the production cost and time [4].

They are many model-based approaches addressing deadlock problem in manufacturing systems. The key benefits of using models at runtime to detect deadlocks in manufacturing systems is that models can provide a rich semantic base for decision-making related to deadlock problem. However, to the best of our knowledge, none of the previously proposed strategies addresses the deadlock problem in the multi-agent manufacturing systems or in the other words manufacturing systems that have been automated by using agent based software engineering methodologies. In this paper we tackle the problem of deadlock in the multi-agent manufacturing systems analyzed and designed by means of Multi-agents Software Engineering (MaSE) methodology [5]. MaSE is one of the Agent Oriented Software Engineering (AOSE)

1082-3409/09 $26.00 © 2009 IEEE
DOI 10.1109/ICTAI.2009.112

[3] methodologies that uses several UML-like models and diagrams to describe the architecture-independent structure of agents and their interactions. In this paper, we focus on a monitoring technique for deadlock detection in multi-agent manufacturing systems using the MaSE constructs.

The remainder of this paper is structured as follows. The related work and background are described in Section 2. The proposed technique overview in this paper is discussed in Section 3. The input design model to our monitoring technique is introduced in Section 4. Constructing the machine requirement table used for finding the potential deadlocks is expressed in Section 5. An algorithm to extract potential deadlocks from the design models is described in Section 6. The proposed communication protocol for deadlock detection is discussed in Section 7. The case study and the experimental results are discussed in Section 8. Finally, Section 9 concludes the paper.

2. Background and related work

In this section, the background information required for describing the proposed technique in this paper is discussed in Sections 2.1 and 2.2. Then, we discuss the related work on deadlock detection in manufacturing systems and the techniques that they used in Section 2.3.

2.1 Deadlock in automated manufacturing systems

An automated manufacturing system usually consists of a set of cells, a material handling system connecting the cells, and service centers including material warehouse, tools room, and equipment repair. A cell can be either a machine, inspector, or a load/unload robot. Therefore, an automated manufacturing system can also be defined as a set of machines in which parts are automatically transported from one machine to another for processing. The deadlock problem has been already addressed in the Operating Systems (OS) and Distributed Database Systems (DDS) contexts. In those contexts, 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" [6]. In this paper, we tackle the challenge of deadlock detection on resources which model physical objects (e.g. machines, robots, drives, etc.). In automated manufacturing system, those resources are shared by processes (i.e. agent tasks).

Coffman et al. [7] provided four conditions that must be held for a deadlock to occur: (1) "Mutual Exclusion" which means each resource can only be assigned to exactly one process; (2) "Hold and Wait" in which processes can hold resources 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. In automated manufacturing system, the first three conditions given by Coffman et al. [7] are always satisfied. Agent tasks (i.e. manufacturing processes) use resources (i.e. machines) in an exclusive mode, they hold resources while waiting for the next resources specified by their operation sequence, and resources cannot be forcibly removed from the parts utilizing them until operation is completed. Therefore, a deadlock can only occur if the fourth condition (i.e. circular wait) is held [4]. In this paper, we tackle the problem of deadlock detection by finding the situations that can potentially lead to a circular wait. An example of system deadlock involving four machines is illustrated in Figure 1. In this figure, the moving parts (parts 1 and 2) are shown by white circles while the machines (machines *A*-*D*) are shown by grey rectangles. Part 1 on machine *A* has to reach machine *D* and part 2 on machine *D* has to reach machine *A*. A deadlock defiantly occurs between machines *B* and *C*, when the part 1 is moving to machine *C* and part 2 is moving to machine *B*.

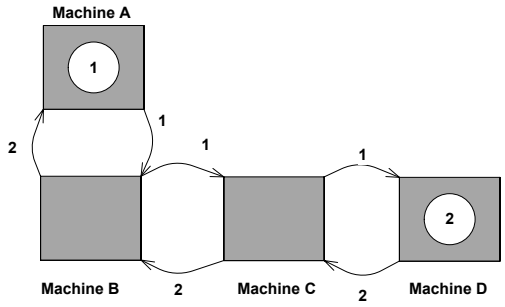


Figure 1 : A deadlock situation involving manufacturing machines and parts

2.2 Agent Based Development Methodology: MaSE

Artificial Intelligence (AI) techniques have significantly impacted manufacturing systems. Many automated manufacturing system application ranging from material handling and assembly controllers to long-term planning have been developed by means of MAS [8] technologies. As a result of the growing demand in MAS for industrial applications, many AOSE methodologies such as MaSE have been evolved to assist the development of agent-based applications [3].

MaSE uses several models and diagrams driven from the standard UML [9] to describe the architecture-independent structure of agents and their interactions [5]. 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 the 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 [5]

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	Task Diagram, 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 this paper, we propose our deadlock detection technique by using the MaSE task diagrams as the input model to our monitoring technique. We propose that in the multi-agent manufacturing systems, MaSE task diagrams are used for modeling the defined production and assembly jobs. More details on MaSE task diagrams as the input to our monitoring technique and using them for deadlock detection in automated manufacturing systems designed by MaSE methodology are discussed in subsequent sections.

2.3 Deadlock detection strategies for automated manufacturing systems

Four main strategies addressing deadlock issues in manufacturing systems are prevention, detection, recovery, and avoidance methods. These strategies can be further categorized based on the model used to describe the AMS and, in particular, the interaction between jobs and resources. Three modeling methods are usually used: graph-theoretic, automata, and Petri nets (PN). The graph-theoretic approaches such as [10] are simple and intuitive solutions appropriate for describing the interactions between jobs and resources. This allows an efficient deadlock depiction even in complex Resource Allocation Systems (RAS) and allows the derivation of deadlock detection and avoidance strategies. Finite state automata can formally model the automated manufacturing systems behavior and have been used in establishing new deadlock avoidance control techniques such as [11]. Petri nets (PN) also have been used extensively by researchers for modeling automated manufacturing systems (e.g.[12]) and to develop suitable deadlock resolution methods (e.g.[13]). But to the best of our knowledge, despite the growing demand of MAS applications in design, analysis, and development of automated manufacturing systems [8], there is no deadlock resolution or detection strategy based on the models created for multi-agent manufacturing systems based on the one of AOSE methodologies [3] such as MaSE [5]. In the following sections, we propose a monitoring technique for deadlock detection in multi-agent manufacturing systems based on the models constructed during MaSE methodology analysis phases.

3. The technique overview

An overview of the proposed model-based monitoring technique in this paper is illustrated in Figure 2. As Figure 2 shows, the proposed technique has three main phases which are briefly described in this section of the paper. During the first two phase of the proposed technique (phase 1 and phase 2), the situations that can potentially lead to a deadlock at runtime (potential deadlocks) are extracted from the design models. Then, in phase 3, the manufacturing system is instrumented with a communication protocol which can use the potential deadlock information extracted from the design model in phase 1 and phase 2 and detect runtime deadlocks by comparing the runtime events with the potential deadlock scenarios.

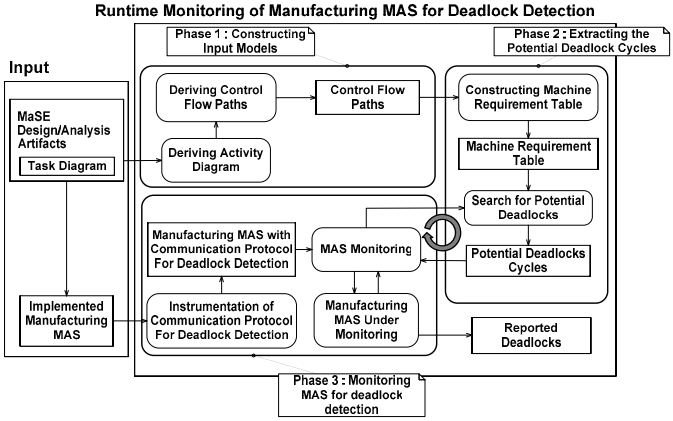


Figure 2 : Technique overview

The input to the proposed deadlock monitoring technique is the agents' task diagrams built during the analysis and design of the MaSE methodology. These diagrams illustrate the activities performed by several cells (e.g. machines and robots) inside the multi-agent manufacturing system. During the steps in the first phase, each MaSE task diagram, a UML-like statechart diagram is converted to a UML activity diagram, from which Control Flow Paths (CFP) are derived afterwards (Section 4). Those CFPs are used in phase 2 to extract the potential deadlock information by means of a dedicated search algorithm (Section 6). This extraction is performed by the "search for potential deadlock" unit upon receiving the request from the monitoring unit in phase 3. In phase 3, the implemented manufacturing MAS designed by MaSE methodology is instrumented with a deadlock detection

communication protocol (Section 7). The communication protocol uses the extracted potential deadlock information (found by "search for potential deadlock" unit in phase 2) to propagate deadlock detection messages among the agents which may involve in a deadlock situation. This way the communication protocol can find out if there is any match between the current running event in the system and the found deadlock scenarios. Therefore, there is a feedback loop between phase 2 and phase 3 at runtime to make sure all deadlocks are detected efficiently.

4. The input task model

In MaSE, a task is a structured set of communications and activities, represented by a UML-like state diagram which consists of states and transitions [5]. 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 [9]. However in MaSE [5], this diagram is used to represent the behavior of a task associated to an agent. In a MaSE task diagram, states contain activities that represent internal reasoning, reading a percept from sensors, or performing actions via actuators. Multiple activities can be in a single state and are performed in an un-interruptable sequence. Once in a state, the task remains there until the activities' sequence is completed [5]. In a multi-agent manufacturing system, the task diagram is used for representing the flow of manufacturing activities. An example of a MaSE task diagram created for a specific task in a multi-agent manufacturing system is shown in Figure 3.

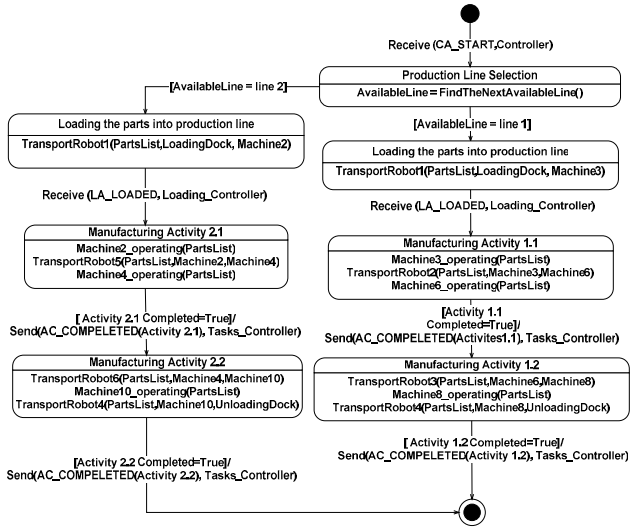


Figure 3: An example input model: A MaSE task diagram for a manufacturing task

As the task diagram shows, this task is able to perform its job through two alternative production lines (flow paths). When the task is initiated by the controller, the task finds the next available production line and afterwards makes the transportation robot to load the parts into that line. After performing each set of activities in each state, the task communicates with the controller agent through the "send" and "receive" messages and informs it about the task status and its current state.

In addition to the information regarding the flow of activities inside each task diagram, the task diagram has the information regarding the sequence of resources (i.e. machines and robots) that are required by the task during the execution of its activities. On the other hand, in this paper we tackle the problem of deadlock on the resources (i.e. machines and robots) shared by agents' tasks. Therefore, in the proposed technique in this paper, we use MaSE task diagrams for extracting the information regarding the required resources by each agent's task. This information is obtained during the steps in phase 1 of the proposed technique in this paper (See Figure 2) by converting the task diagrams into UML activity diagrams. Figure 4-(a) shows the constructed activity diagram based on the task model provided in Figure 3. 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 [14]. As it is mentioned in UML 2.0 (Section 12.1 of [9]), the UML activity diagrams are commonly called Control Flow Graph (CFG). By analysis of the control flow paths (CFP) in the constructed CFG or activity diagram the resource requirement flow of the task can be extracted. The procedure is described in more details in the remainder of this section.

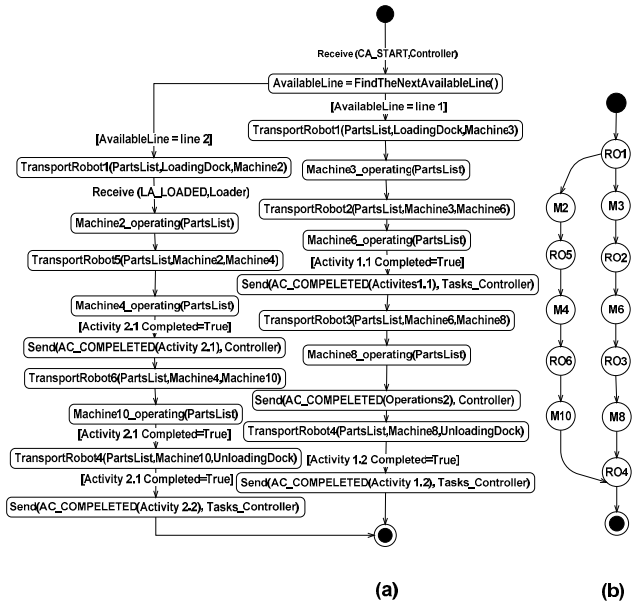


Figure 4 : (a) Constructed activity diagram based on the Task Diagram (b) Resource Requirement Flow Model

Each state in MaSE task diagram contains the activities representing internal behavior of the task when it is on that state. Since the order of task's states is presented in the MaSE task diagram by the directed transition arrows, the activities within the task, the sequence of their execution and their execution conditions can be easily driven from the states and the transition arrows and then they are imported into an new activity diagram.

Furthermore, the transition protocol (i.e. "send" and "receive" message communications) in MaSE task diagram uses the syntax of trigger [guard] / transmission and the trigger and transmission are limited to send and receive messages [5] (See Figure 3). Therefore, in the newly constructed activity diagram, each trigger and [guard] form the MaSE task diagram should be considered as condition of the transition from the activities in source state to the activities in destination state. Also, transmission should be considered as a new activity executed after all activities listed in the source state (See Figure 4-(a)). As a result, using the derived activities from the states and the transitions among them, the corresponding activity diagram for a MaSE task diagram is created in Figure 4-(a). Using the sequence of activities in the constructed activity diagram (i.e. CFPs), the resources used by each tasks activity can be extracted and imported into a Resource Requirement Flow Model (Figure 4-(b)). In the Resource Requirement Flow Model presented in Figure 4-(b), the resources (i.e. machines and transport robots) used during the execution of each CFP are presented. Each manufacturing machine is shown by M_i and each transport robot by RO_i. Between the manufacturing machines and the transport robots resource types, only the manufacturing machines are selected and considered for tackling the deadlock problem in this paper. This selection is based on the assumption that the transport robots are only used for carrying the parts and they are not directly involved in the manufacturing jobs such as assembly or modification. Therefore, they can be forcibly taken from the tasks by the others agents' requests and be involved in other tasks by providing them the storages that they can move the carrying parts into them temporarily. In this case, they do not satisfy the Coffman et al. [7] conditions for a deadlock situation and they must be ignored for the deadlock detection technique proposed in this paper.

5. Machine requirements table

We use the resource requirement information obtained from each CFP in the activity diagram to find the potential deadlocks inside the design models. But before using this information for extracting the deadlock information, we organize the gathered the resource requirement information from the activity diagram into a table called machine requirement table. As discussed in Section 4, among all the resources types extracted from the input model, only the manufacturing machine are considered for the process of deadlock detection in this paper. Therefore, each column in machine requirements table represents the Sequence of Required Machines (SRM_i) by one CFP_i . Formally, The SRM is defined as below:

$$SRM_i = < M_j | M_j \text{ is the jth required Machine of the } CFP_i >$$

Figure 5 shows an example of a machine requirement table created for the case study in this paper. The first two columns shows the SRMs for the two CFPs extracted from the constructed activity diagram in Figure 4-(a).

Agent #	$Agent_1$		$Agent_2$	$Agent_3$		$Agent_4$	
Task #	$Task_1$		$Task_3$	$Task_4$	$Task_5$	$Task_6$	$Task_7$
CFP #	$CFP_{1,1}$	$CFP_{1,2}$	$CFP_{3,1}$	$CFP_{6,2}$	$CFP_{5,1}$	$CFP_{6,1}$	$CFP_{7,1}$
Required Machines (Resources)	M_3	M_2	M_1	M_{11}	M_2	M_5	M_{11}
	M_6	M_4	M_4	M_9	M_4	M_4	M_8
	M_8	M_{10}	M_7	M_7	M_3	M_3	
			M_{10}	M_5			

Figure 5 : Machine Requirement Table for manufacturing tasks

6. Extracting the potential deadlocks

The constructed machine requirements table in previous section is used in this section for extracting potential deadlocks in the system. The deadlocks found from the design models in this step are called potential deadlock cycles since they can potentially lead to a deadlock situation at runtime. In this technique, a $Task_1$ is said to be dependent on another $Task_k$ if there exists a sequence of tasks such as $Task_1$, $Task_2$, …, $Task_k$ where each task in sequence is idle and waiting for a resource held by the next task in the sequence. If $Task_1$ is dependent on $Task_k$, then $Task_1$ has to remain in idle status as long as $Task_k$ is idle. $Task_1$ is deadlocked if it is dependent on itself or on a task which is dependent on itself. The deadlock can be extended to a cycle of idle tasks, each dependent on the next in the cycle. This is called a potential deadlock.

Our search technique uses the machine requirements table and searches for the tasks' combinations that can potentially lead to a deadlock cycle. These cycles and the participant agents in it are candidates for potential deadlock cycles and are used for the deadlock detection message propagation discussed in Section 7. For describing the procedure of our search technique in this paper, an illustrated example is provided and shown in Figure 6. The provided example in Figure 6 shows three tasks of a multi-agent manufacturing system. Each task runs a CFP from its activity diagram (i.e. CFG). Associated to each CFP is a SRM (SRM_1, SRM_2, and SRM_3) representing the sequence of required resource by task on that CFP. The proposed search technique works as

follow: starting from $Task_1$, the search technique assumes that it holds its first requested machine in the SRM_1 which is M_1 and adds M_1 to a SofarTraversed list (i.e. SofarTraversed = <M_1>). Therefore, the next machine that the $Task_1$ will request for it after holding M_1 is M_2. In this stage, the technique searches inside the other SRMs (SRM_2, SRM_3) to find out if there is any other request for M_2 from the other tasks as well. It finds that $Task_2$ has M_2 as a requested machine in its SRM (SRM_2) and therefore it may compete with $Task_1$ for acquiring M_2. In this stage, the technique assumes that in the worst case scenario, the M_2 has been held by $Task_2$ and the Task1 has to wait for it and then add the M_2 to the SofarTraversed list (i.e. SofarTraversed = <M_1, M_2>). Therefore, if the $Task_2$ is holding M_2 then the next requested machine by $Task_2$ from SRM_2 will be M_3. In this stage, the technique searches the SRM_1 and SRM_3 to find any match for M_3. A match is found in SRM_3 and again the technique assumes that M_3 is held by $Task_3$ and $Task_2$ has to wait for it. M_3 is added to the SofarTraversed list (i.e. SofarTraversed = <M_1, M_2, M_3>). After M_3, the next requested machine by $Task_3$ is M_1. The technique finds a match for M_1 in SRM_1. The technique has to assume that the M_1 is held by $Task_1$ and $Task_3$ has to wait for it. But, the technique finds out that M_1 has been already added in to SofarTraversed and this means that the M_1 has been already traversed by our proposed technique. In this stage, technique finds a cycle of holds and requests which can potentially lead to a deadlock at runtime. The technique reports this cycle as a potential deadlock. This procedure is repeated by initiating the search from all other the SRMs and the resources inside them till the entire potential deadlocks are found. A possible pseudo-code for the technique discussed in this section is presented in Figure 7.

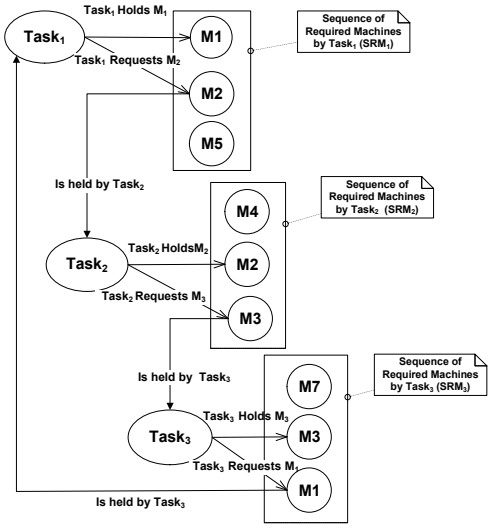


Figure 6 : An example of potential deadlock of tasks

In Figure 7 pseudo-code, first the traversed SRM (SRM_i), held machine (M_i), and requested machine (M_{i+1}) are added into a thus far traversed list called *SofarTraversedList* (lines 3-5). Then, if the newly added items into the *SofarTraversedList* make a cycle with the existing items in it, the created *SofarTraversedList* is printed as a poetical deadlock cycle (lines 8-10). Otherwise, the requested machine (M_{i+1}) is searched in the all other SRMs except the existing SRMs in *SofarTraversedList* to find a match (lines12-19). If any match found, the requested machine (M_{i+1} or *Next*) is assumed as a held machine and the next required machine in the SRM that the match is found in it is assumed as the requested machine (*NextRequiredMachine*) and the same procedure is called again till a cycle is found or all the SRM are traversed without any found match.

```
1  private void Search_Potential_Deadlocks( List  SofarTraveresed , List SRMi , Int  Mi ,Int  Mi+1)
2  {
3    List SofarTraversedList = SofarTraveresed ;
4     Next  =  Mi+1;
5     Add  (SRMi , Mi , Mi+1 )  to SoFarTraversedList
6     if (Mi == Last Requested Set  in SRMi)
7                 return;
8     for (int k = 0; k< SoFarTraversedList.Length-1 ; k++)
9          if (SoFarTraversedList [k]  == Next )
10               Print all the items in SoFarTraversedList as a Potential Deadlocks Cycle ;
11
12    for ( int p=0 ;All other SRMs (SRMp) in the MAS except the SRMi  ; p++)
13               if ( SRMp does not exist in SofarTraversedList )
14                  if ( Next  exists in the SRMp)
15                       {
16                       NextRequiredMachine = Find the required set next set to Next  in SRMp;
17                       Found= true;
18                       Search_Potential_Deadlocks (SofarTraveresed , SRMp , Next ,NextRequiredMachine  );
19                       }
20    if ( Found == false)
21               return;
22  }
```

Figure 7 : Pesudo-code for finding the potential deadlocks

7. A communication protocol for deadlock detection

The potential deadlock cycles extracted by the search technique represents the situation that can lead to a deadlock at runtime. In this section, we proposed a communication protocol which is able to detect deadlocks by monitoring the multi-agent manufacturing system and comparing the running events with the potential deadlock scenarios extracted from the design models. In this deadlock detection protocol, each agent is associated with a set of dependent agents called the "dependency set". Each agent identifies its dependency set by using the potential deadlock information extracted from the MaSE task diagrams (i.e. the design input models). The dependency set for each agent is the set of agents involved in a potential deadlock cycle (extracted by dedicated search algorithm described in Section 6) with that agent. Whenever an agent is suspected to be involved in a deadlock situation (after spending a defined amount of time in an idle mode), the monitoring unit provides the current running task and its CFPs for that agent to "search for potential deadlocks" unit in phase 2. The "search for potential deadlocks" unit extracts the possible potential deadlocks and provides the involved agents as the dependency set for that agent.

Assuming each CFP is running by an agent in multi-agent manufacturing system, an agent in a dependency set can change its status from idle to active upon receiving any message from one of the other members of its dependency set. A nonempty set of agents are considered as deadlocked if all agents in that set are permanently idle. An agent is called permanently idle, if it never receive a message from its dependency set to change its status. An agent can determine if it is deadlocked by initiating a deadlock detection query messages to its dependency set when it enters the idle state. If it figures out that it won't receive any message from its dependency set to change its status, it declares itself as deadlocked (permanently idle). Therefore, an agent A_i does not declare itself as deadlocked if and only if it initiates query messages to all the agents in its dependency set and receives positive reply to every query that it has initiated. Upon receiving a query by an idle agent in dependency set, it forwards the query to its own dependency set too if it has not done already. Figure 8 shows an example of deadlock detection conversations among agents. The query conversation messages are shown by continues arrows and the replies are shown by dashed arrows. In this example, Agent 1 initiates two deadlock detection queries to its dependency set: Agent 2, Agent 3, and Agent 4. Agent 2 can reply to the query immediately by informing Agent 1 about its state since it is not involved in any other dependency set. Agent 3 and Agent 4 are involved in other dependency sets and they pass the query to their own dependency set before informing Agent 1 about their status. Agent 3 and Agent 4 answer the Agent 1's query upon receiving the replies from their dependency set. Based on the replies received, Agent 1 can identify whether its status can be changed from idle to active or not.

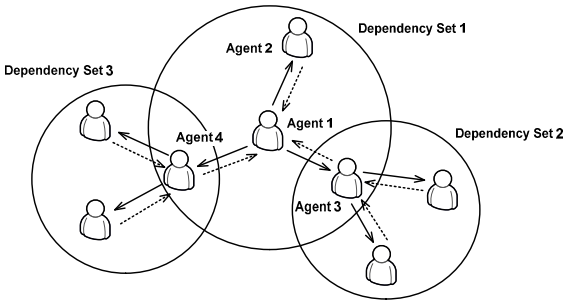


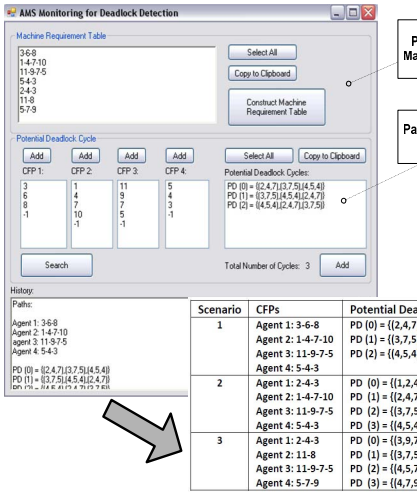
Figure 8 : Deadlock Detection Conversation

8. Case study

We performed an experiment for showing the effectiveness of our technique based on a multi-agent manufacturing system with capability of running four different CFPs concurrently by four agents at each time. Figure 9 provides a snapshot from the developed tool used for the experiment in this paper. This tool is the implementation of phase 1 and phase 2 of the proposed technique in this paper (See the technique overview in Figure 2).

This tool is able to construct the Machine Requirement Table's columns by extracting the CFPs from the MaSE task diagrams (pane 1 in Figure 9). Whenever an agent is suspected to be in a deadlock situation (after spending an adjustable amount of time in an idle mode), the monitoring unit informs the "search for potential deadlocks" unit about the currently running CFPs in the system. The "search for potential deadlocks" unit implemented in the Figure 9's tool is able to extract the potential deadlock situation and the participated agents afterward (pane 2 in Figure 9). The table below the tool snapshot in Figure 9 shows the generated potential deadlock cycles for three different deadlock situations. In the first scenario, the participant agents in the deadlock are Agent 2, Agent 3, and Agent 4. Therefore, the query messages are initiated to these agents. In the second scenario, all the agents are involved. In the third scenario, Agent 1 and Agent 2 can perform their tasks without any conflict, while Agent 3 and Agent 4 can lead to a deadlock. Therefore, the deadlock detection query messages are propagated to the latter set as the dependency set of each other.

Furthermore, for showing the effectiveness of deadlock communication protocol in phase 3 (Section 7), another tool has been developed for simulating the agents' behavior in a manufacturing system. This tool is able to show each agent behavior by simulating the resource holds and requests during the execution of each CFP. The monitoring module instrumented with the deadlock detection communication protocol has been also embedded to the developed simulator.



Scenario	CFPs	Potential Deadlocks	Participated Agents
1	Agent 1: 3-6-8 Agent 2: 1-4-7-10 Agent 3: 11-9-7-5 Agent 4: 5-4-3	PD (0) = {(2,4,7),(3,7,5),(4,5,4)} PD (1) = {(3,7,5),(4,5,4),(2,4,7)} PD (2) = {(4,5,4),(2,4,7),(3,7,5)}	Agent 2, Agent 3, Agent 4
2	Agent 1: 2-4-3 Agent 2: 1-4-7-10 Agent 3: 11-9-7-5 Agent 4: 5-4-3	PD (0) = {(1,2,4),(2,4,7),(3,7,5),(4,5,4)} PD (1) = {(2,4,7),(3,7,5),(4,5,4)} PD (2) = {(3,7,5),(4,5,4),(2,4,7)} PD (3) = {(4,5,4),(2,4,7),(3,7,5)}	Agent 1,Agent 2, Agent 3, Agent 4
3	Agent 1: 2-4-3 Agent 2: 11-8 Agent 3: 11-9-7-5 Agent 4: 5-7-9	PD (0) = {(3,9,7),(4,7,9)} PD (1) = {(3,7,5),(4,5,7)} PD (2) = {(4,5,7),(3,7,5)} PD (3) = {(4,7,9),(3,9,7)}	Agent 3, Agent 4

Figure 9 : Monitoring multi-agent manufacturing system for deadlock detection

Whenever an agent is suspected to be in a deadlock situation at runtime, the monitoring unit uses the potential

deadlock information provided by the tool in Figure 9 and initiates the deadlock detection query messages to the participated agents. If it finds out that that the current situation matches the potential deadlock scenario provided by the search unit, it reports a deadlock. Identifying the dependency set at runtime based on the design artifacts helps the monitoring module to initiate the query messages to the participating agents in the deadlock only and not all of the agents in the system. As an example in the third scenario, the deadlock detection query messages could be initiated to all the agents in the multi-agent manufacturing system including Agent 1, Agent 2, Agent 3, and agent 4. This could result to communication traffic overhead specifically in the multi-agent manufacturing with a higher number of agents. By using our technique, the deadlock query messages are only initiated to half of them which are Agent 3 and Agent 4. This reduces the internal message communication traffic of the system to a large extent.

9. Conclusion

A monitoring technique for deadlock detection in manufacturing MAS was proposed. The MaSE task diagram is used for modeling the production jobs in a multi-agent manufacturing system. Defined tasks are performed by the agents inside a multi-agent manufacturing system. Potential deadlocks are extracted using the CFP derived from the MaSE task diagram and a dedicated search algorithm. Using the extracted potential deadlocks information, each agent is able to find its dependency set based on the agents involved in a potential deadlock situation. Whenever an agent is suspected to be in a runtime deadlock situation, it propagates a deadlock detection query messages to its dependency set to figure out if it is permanently idle. Finally, a tool was developed for performing an experiment on a multi-agent manufacturing system with capacity of running four CFPs concurrently. The experiment results show that the technique is also able to reduce the internal message communication traffic of the system by limiting the number of agents that the deadlock detection query messages should be initiated to.

10. Acknowledgement

The authors were supported by discovery grants from NSERC. Vahid Garousi was further supported by an Alberta Ingenuity New Faculty Award no. 200600673.

11. References

[1] K. Kumaran, W. Chang, H. Cho, and R. A. Wysk, "A structured approach to deadlock detection, avoidance and resolution in flexible manufacturing systems," *International Journal of Production Research,* vol. 32 (10), pp. 2361-2379, Oct.1994.

[2] M. R. Genesereth and P. K. Ketchpel, "Software agents," *Commun. ACM,* vol. 37 (7), pp. 48-53, 1994.

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

[4] M.P. Fanti and M. Zhou, "Deadlock control methods in automated manufacturing system," *IEEE Transactions on Systems, Man and Cybernetics,* vol. 34 (1), pp. 5-22, Jan. 2004.

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

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

[7] E. G. Coffman, M. J. Elphick, and A. Shoshani, "System deadlocks," *ACM Comput. Surv.,* vol. 3, pp. 67–78, June 1971.

[8] Weiming Shen, D. H. Norrie, and Jean-Paul Barthès, *Multi-agent Systems for Concurrent Intelligent Design and Manufacturing*. London: Taylor & Francis Press, 2001.

[9] Object Management Group (OMG), "UML 2.1.2 Superstructure Specification," November 2007.

[10] N. Z. Gebraeel and M. A. Lawley, "Deadlock detection, prevention, and avoidance for automated tool sharing systems," *IEEE Trans. Robot. Automat,* vol. 17, pp. 342–356, June 2001.

[11] S. A. Reveliotis and P. M. Ferreira, "Deadlock avoidance policies for automated manufacturing cells," *IEEE Trans. Robot. Automat.,* vol. 12, pp. 845–857, 1996.

[12] A. A. Desrochers and R. Y. Al-Jaar, *Applications of Petri Nets in Manufacturing Systems*. New York: IEEE, 1995.

[13] F. Chu and X. Xie, "Deadlock analysis of Petri nets using siphons and mathematical programming," *IEEE Trans. Robot. Automat.,* vol. 13, pp. 793–804, Dec. 1997.

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