Deadlock detection in agent-based virtual knowledge communities

Ogunleye Gabriel Opeyemi [1] Adewale O. S.[2] Adetunmbi A.O.[2]
Alese B.K.[2] Ogunde A.O.[1]
[1]Department of Mathematical Science (Computer Science Programme), Redeemer's University, Redemption Camp, Mowe, Ogun State, Nigeria
[2]Department of Computer Science, School of Science, Federal University of Technology, Akure, Nigeria

ABSTRACT. In attempt to develop an intelligent agent based model for deadlock detection in Virtual Knowledge Communities (VKC), shadow and detector agents were used to facilitate the detection process in the design. A mathematical model based on the design for deadlock detection and resolution in multi-agent communities was developed. The prototype implementation was carried out using Java platform technology. Experimental result is satisfactory in addressing the issue of deadlock problems in knowledge sharing in agent-based VKC.
KEYWORDS. Knowledge Management; Mobile Agent; Virtual Knowledge Communities; Deadlock Detection; Deadlock Resolution

Introduction

Knowledge is known as an important strength in any organization. The purpose of any organization is to make decision or to support decision making on series of issues. Therefore, knowledge has to be shared among different people in the organization so as to promote the growth of the company. Most knowledge workers presently spend most of their time working virtually through the information and communication technologies, but have not taken note of the problem involved. Even the growth of any country is centered on the knowledge acquired. There are even claims that

knowledge can be found in ontology. Ontology had its root from Greek philosophy. Knowledge can also be linked to the area of artificial intelligence (AI).

However, knowledge representation was one of the main constituents of AI. Multi-agent systems have also been used to accomplish the tasks in distributed AI. Virtual knowledge community is a virtual place where agents can meet, communicate and interact among themselves [PJ09]. Virtual Knowledge Communities are a way to allow the users to build their own knowledge sources, to design them as different threads in a particular methodology. The increasing trend among commercial firms for knowledge sharing operations, and the consequent need by some organization for knowledge sharing among their workers have placed heavy demands on knowledge sharing activities in VKC. Compounding these difficulties, distributed knowledge processing has arrived as the solution to incremental system growth. Such contemporary systems exhibit a high degree of resource and knowledge sharing, a situation in which deadlock is a constant threat. Deadlock occurs when processes holding some resources request access to resources held by other processes in the same set [Sin89].

A lot of research works had been done proposing different schemes and algorithms on knowledge sharing and management in VKC [BM02]. Deadlock detection problem in VKC has not been given much attention in the research community. As agents are requesting and sharing knowledge in VKC, deadlock could arise. Consequently, this paper will center on deadlock detection and recovery in VKC, thereby resulting into effective communication and sharing of knowledge among the agents. The problem of deadlock can only be broken by the involvement of some external agency [IM80]. The advent of mobile agents, which are intelligent programs endowed with the property of mobility along with other properties like autonomy, intelligence, adaptability etc seems to be a promising panacea to this problem. Therefore, shadow agents would be used in our model as part of deadlock detecting process. The shadow agents maintains one part of the Wait-for-Graph (WFG) and searches for circles that in the resource model that necessarily constitute deadlocks. This paper therefore presents a framework for knowledge sharing activities in multi-agent systems that is free against any deadlock problem. The rest of the paper is organized as follows. In section 1, Knowledge Management, mobile agents, virtual knowledge communities, deadlock are reviewed. In section 2, our deadlock detection method is introduced based on intelligent shadow and detector agents. Section 3 presents the case study/Experimental set-up; section 4 gives some results and discussions while conclusions are drawn in section 5.

1 Review of Related Literature

1.1 Knowledge Management

Knowledge is now recognized as the driver of productivity and economic growth, leading to a new focus on the role of information, technology and learning in economic performance [OEC96]. KM (Knowledge Management) in its broadest sense is a conceptual framework that encompasses all activities and perspectives required to gain an overview of, deal with, and benefit from the corporation's knowledge assets and their conditions. It pinpoints and prioritizes those knowledge areas that require management attention. It identifies the salient alternatives and suggests methods for managing them, and conducts activities required to achieve desired results [Wii93]. The initiation and spread of the internet has taken the information age to a new level of complexity. The information society now has so much information at its disposal that it is more important than ever to find effective techniques for managing the optimal distribution of this information, such that individuals are not overwhelmed with meaningless data. The objectives of knowledge management are well known; to improve the reuse of the knowledge within the processes of a system, by reducing the distance between tasks and generalized knowledge bases, and increasing accessibility to resources. In recent years many organizations have adopted knowledge management techniques that focus on building large, expensive, centralized knowledge management systems based on the standardization of the syntactical and semantic representations of all of the knowledge in the organization.

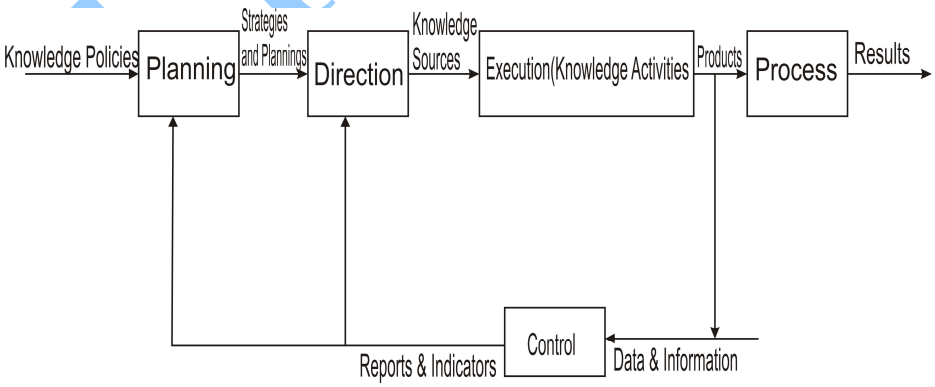

Figure 1. Knowledge management model

1.2 Mobile Agents

A mobile agent is a kind of software program that can migrate from one host to another in a heterogeneous network [HC07]. Also known as travelling agents, these programs will shuttle their being, code and state among resources. They are network nomads that act as personal representative, working autonomously through networks. They are able to visit network nodes directly using available computing power and are not limited by platform. The technology has become an alternative approach for the design and implementation of distributed systems to the traditional Client/Server architecture.

Mobile agents can migrate from one system to another during their execution and communicate amongst one another, clone, merge and co-ordinate their computations. Mobile agents are autonomous agents in the sense that they control their relocation behavior in pursuit of the goals with which they are tasked. Main fields of application for mobile agents are information retrieval on the www, distributed database access, parallel processing, automation of electronic marketplaces and others. Mobile agent frameworks are currently rare, due to the high level of trust required to accept a foreign agent into one's data server. However with the advances in technologies for accountability and immunity, mobile agent systems are expected to become more popular in the future.

1.3 Virtual Knowledge Communities

Virtual communities are becoming increasingly popular, particularly on the Internet, as a means for like-minded individuals to meet other individuals they can learn to trust and to share and gain access quickly and efficiently to the information they are mostly interested in. The concept of a community of practice or a community of interest can be supported in a virtual community in order to bring the appropriate parties together and to share their knowledge [PJ09]. The advantage of this is that the members of a community centered on one specific topic or practice will only be presented with knowledge from domains they are, or at least are relatively likely to be, interested in. This knowledge needs not be something they have specifically asked/searched for. Many virtual community applications already exist on

the Internet. Some are using agents in various forms as part of the back office system.

[PJ09] proposed an approach that extends the abstraction of an agent, such that it acts within the system, searching for or delivering knowledge within other agents and through communities. With such a model, agents can choose to join, leave, create and destroy a community, they can ask for information and send information to the community, and they can be member of several communities simultaneously. Virtual Knowledge Community (VKC) was called the virtual place where agents can meet, communicate and interact among themselves. Basically, a VKC is centered on a topic, corresponding to a domain of interest for which the interested agents have joined this community. This notion allows an increased availability of data and knowledge within the various communities.

A community consists of a domain of interest (a knowledge cluster), a leader (an agent), a policy and an unspecified number of member agents. Also [Fer97] gives the following definition for a multi-agent system: "a system composed of a population of autonomous agents which cooperate with each other to reach common objectives, while simultaneously each agent pursues individual objectives." We can thus see a Multi-Agent System as a system in which autonomous agents can communicate, exchange their individual knowledge and cooperate in order to solve complex problems and to achieve collective or individual goals [HS99]. This is the natural and most general available model for an agent-based knowledge sharing system.

Hence, knowledge sharing is now a common practice on the internet today and if not handled with care especially when the agents and communities increased, agents will become deadlocked in the process of sharing or exchanging resources. Deadlock is a constant threat in terminal oriented systems. This is called for comprehensive study of deadlock.

1.4 Deadlock

Deadlocks arise when members of a group of processes which hold resources are blocked indefinitely from access to resources held by other processes within the group. When no member of the group will relinquish control over its resources until after it has completed its current resource acquisition, deadlock is inevitable [IM80]. There are two types of deadlock, resource deadlock and communication deadlock. In resource deadlocks,

processes make access to resources (for example, data objects in database systems, buffers in store and forward communication networks). A process acquires a resource before accessing it and releases it after using it. A process that needs resources for execution cannot proceed until it has acquired all those resources. A set of processes is resource deadlocked if each process in the set requests a resource held by another process in the set.

In communication deadlocks, messages are the resources for which processes wait. A set of processes is communication deadlocked if each process in the set is waiting for a message from another process in the set and no process ever sends a message. To handle deadlocks in distributed systems, one can try to adopt approaches known from centralized systems, i.e. prevention, avoidance and detection with recovery.

In deadlock prevention, a deadlock state is made unreachable by violating one of the necessary conditions for deadlock. For example, imposing a fixed order in which resources are acquired, such as in "monotone locking", violates the condition for a cyclic dependency among resources. This strategy, however, enforces some burden on the programmer, and often a more important concern in embedded systems. Prevention is the process of restraining system users so that requests leading to deadlock never occur. Most proposals for prevention require each process to specify all needed resources before transactions begin.

Deadlocks can be prevented in various ways, including requesting all resources at once, preempt or displace resources held, and ordering resources. The most elementary way of preventing deadlock is to outlaw concurrency, but this normally result into poor resource utilization and is not consistent with current system design philosophies. Another method requires that all resources be acquired before processing starts. Such a scheme is inefficient, since resources held may be idle for a long time but works well for processes which perform a single burst of activity.

Certain other prevention methods require a blocked process to release resources requested by an active process. For example, when a process requires more main memory than is currently available, it becomes blocked. Afterwards, the process is swapped to secondary storage by preempting its memory for use by active process. The blocked process is swapped back only when the entire, larger quantity of memory is available.

Deadlock detection methods assume that all resource requests will be granted finally. A periodically invoked algorithm analyses present resource allocations and remaining requests to check if any processes or resources are deadlocked. If a deadlock is detected, the system must recover

as soon as possible by preempting resources from affected processes until the deadlock is broken. Detection-scheme overhead includes not only the run-time cost of the algorithm but also the potential losses inherent in preempting resources. Since no action is done until a deadlock arises, resources may be held idle by blocked processes for long periods of time.

According to [Sin89] there are three types of algorithms for deadlock detection in distributed systems: centralized, distributed, and hierarchical algorithms. In centralized algorithms, the global state of the system is known and deadlock detection is simple. In distributed algorithms, the problem of deadlock detection is more complex because no site may have accurate knowledge of the system state. In hierarchical algorithms, sites are arranged in a hierarchy, and a site detects deadlocks involving only its descendant sites.

In avoidance method, a resource request is granted only if at least one way left for all the processes to complete execution. A commonly used deadlock avoidance algorithm used in operating systems is the banker's algorithm which was proposed by Dijkstra to take care of single resource type.

1.5 Agent-Based Deadlock Detection Systems

[Ash00] described a distributed deadlock solution for use in mobile agent systems. The author suggested the properties for the proposed system which are locality of reference, topology independence, fault tolerance, asynchronous operation and freedom of movement. The presented technique proposes dedicated agents for deadlock initiation, detection and resolution. These agents are fully adapted to the properties of a mobile agent environment.

[J+04] developed a framework, called MAEDD (Mobile Agent Enabled Deadlock Detection), for distributed deadlock detection using mobile agents. In MAEDD, mobile agents are dispatched to collect and analyze deadlock information distributed over the network sites, detect deadlock cycles in the system, and then resolve the deadlocks. A prototype implementing the proposed framework was developed using IBM's Aglets. The performance of the mobile agent enabled approach was evaluated against traditional message-passing based solutions. Preliminary performance evaluation results indicated that MAEDD can detect deadlock faster than massage-passing solutions, with increased network traffic.

[NVB08] introduced a methodology for efficient monitoring of Multi Agent System (MAS) to detect resource and communication deadlocks. The authors constructed a behavioral model of 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, they adapted and refine existing resource and communication deadlock detection techniques to the context of MAS. In order to efficiently detect deadlocks and only involve the needed agents in the deadlock detection mechanism, their technique relies on identifying the dependency set for each agent They instrumented the source code of the system with specific instructions in terms of deadlock detection techniques to enable runtime deadlock detection.

3 Deadlock Detection Model

This describes the design of the main concepts upon which the distributed deadlock detection in virtual knowledge communities are developed. The resource represents the knowledge that the agents in a community are sharing. In this approach, shadow agents maintain a part of the wait-for graph on behalf of the community agent. During the deadlock detection process, shadow agents continually check for deadlocks in the global wait for graph and create deadlock detection agents to build the global graph.

For explanation simplicity in this scenario, it is assumed that each resource is associated with an agent. Each agent holds one resource and requests for acquiring the next required resource set. The required resource may have already been acquired by another agent and the requestor has to wait for that resource. In this 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, A_3,…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.

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 As, each dependent on the next in the cycle. Therefore, the deadlock detection approach is to declare the existence of that cycle. The algorithm as

well as the model to represent the framework is represented in figure 2 and figure 3 respectively.

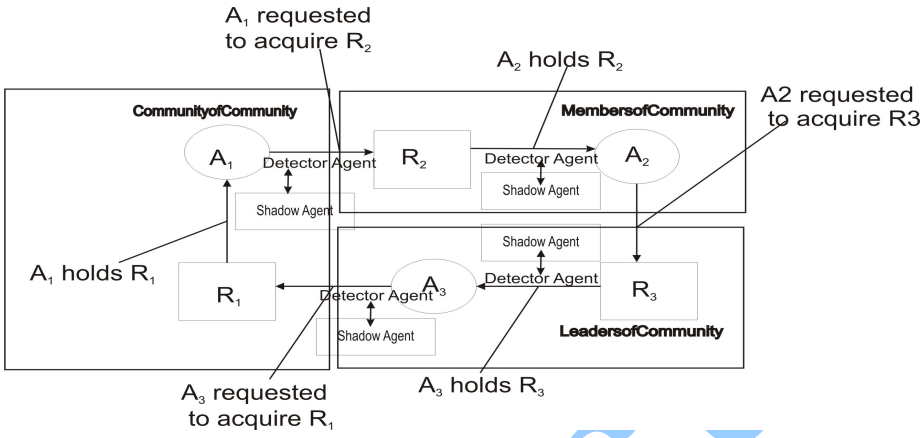

Figure 2. Deadlock Detection in VKC

The description of the behavioral parameters; is stated thus; followed by the mathematical model of the deadlock detection model in VKC.

L = the number of concurrently processing local agents per community (no requirement to access resource at the other communities to complete successfully);

D = the number of concurrently processing distributed agents (D_1,D_2…D_k) per community (either originated at this community and require access to resource at another community, or vice-versa),

A = the total number of concurrently processing agents in a given community,

k = the probability that an agent experiences a lock-wait any time while processing a given resource (Not worry about when the lock-wait occurs, but assume that all waits will be seen by the deadlock detector),

E = expected number of global edges per community.

If agent A_1 is locally dependent on itself or on a set of idle agents all in same community
Then declare A_1 is deadlocked
Else
for all agents(in parallel) such as A_1 which want to acquire resource R (internally or externally) which is held by another agent A_2 in another community,
Send shadow agent(detector agent) to community of agent A_2
End for
End if
A community on receiving a detector agent
If A_2 is idle, and it has not replied to all requests of A_1 recently (after a defined time T), and A_2 has not known that A_1 is dependent on it
Then
Declare that A_1 is dependent on A_2
If R_r is one of the resources which are held by A_2
Then declare that A_2 is deadlocked
Else
For all agents (in parallel) such as Agent A_k which want to acquire resource R_r (internally or externally) which is held by another agent A in another community
Send shadow agent (detector agent) to community of Agent A
End for
End if
End if

Figure 3. Algorithm for deadlock detection in VKC

The lock-wait which are mostly interested are those that form potential global cycles (this requires that there is a situation of $D_1 \rightarrow \ldots \rightarrow D_k$). The probability there is an agent representing a distributed agent from the class of active distributed agent (D_1) is in lock-wait for a resource held by a distributed agent (D_2) can be expressed by

$$D_1 \rightarrow D_2 = D_1 * k * D_2/A$$

Since it must not only wait for a resource, but for a resource owned by one of the distributed agents that are waiting to receive, out of all agents.

The probability that an agent representing a distributed system in the class of active distributed agent (D_1) is in lock-wait for a resource held by a local agents can be expressed by

$$D_1 \rightarrow L = D_1 * k * L/A$$

Since it must not only wait for a resource, but for a resource that is owned by a member of the set of local agents, out of the set of all agents.

The model considers only wait for relationships of the form $D_1 \rightarrow \ldots \rightarrow D_k$ (where there is a string starting with a D_1, and ending with a D_k, with the possibility of one or more local agents in the wait-for string), the probability for a specific local agents waiting for any D_1 is required:

$$(\text{a specific L}) \rightarrow D_1 = k * D_1/A$$

The specific local agents waiting for a D_1 in which is of interest is the one which is being waited for (either directly or indirectly) by a D_2, thereby creating a global edge.

For longer strings, the probability that a specific local agent is waiting for another local agent is

$$(\text{a specific L}) \rightarrow L = k * L/A$$

The probability that an agent represents a distributed agent (D_1) is in lock-wait for a local agent, which in turn, in lock-wait for a resource held by a distributed agent that is waiting to receive can be expressed by

$$D_1 \rightarrow L \rightarrow D_2 = (D_1 * k * L/A) * (k * D_2/A)$$

The probability that an agent represents a distributed agent (D_1) is in lock-wait for a local agent, which is in turn, in lock-wait for a second local agent, which is in turn, in lock-wait for a resource held by a distributed agent that is waiting to receive, can be expressed by

$$D_1 \rightarrow L \rightarrow L \rightarrow D_2 = (D_1 * k * L/A) * (k * L/A) * (k * D_2/A) = D_1 * (k * L/A)^2 * (k * D_2/A)$$

Hence, the above equation shows that there is a global cycle from $D_1 \rightarrow L \rightarrow \ldots D_k$. From this equation, it can be deduced there is actually a deadlock with the relationship between agents waiting for other agents in such a manner that none can proceed.

It then follows that the expected number of global edges that will normally occur in a given community is:

$$E = D_1 * k * D_2/A * (1+(k*L/A)+(k * L/A)^2 + \ldots).$$

The deadlock detection model can simply be represented with the notation above. When deadlock has been detected in a community, then it has to be broken. In this research, deadlock can be broken by the introduction of shadow agents (S_y) which are represented in the next model.

The shadow agents can be represented by the notation S_1, S_2, $S_3 \ldots S_y$. Shadow agents are introduced simply to reduce the set of waiting agents. It grants resource to the agent that has the lowest resource priority. Therefore, from the above equation, assuming the first agent that has the lowest resource priority is D_2. Hence, D_2 is selected. D_2 will now be removed from the set of waiting agents.

$$S_1 \rightarrow (D_1 \rightarrow L \rightarrow L) = S_1((D_1 * k * L/A) * (k * L/A)) = S_1(D_1 * (k * L/A)^2)$$

After a defined time, when agent D_2 has finished using the resources, then the next agent to be selected by the shadow agent is agent L. The next equation now is:

$$S_2 \rightarrow (D1 \rightarrow L) = S_2(D1 * k * L/A)$$

This phenomenon continues until all the waiting agents have been allocated resources. Hence, the string is reduced and deadlock is hereby broken.

The equation is hereby reduced to:

$$S_y \rightarrow D_k$$

It can now be deduced from the above equation that shadow agent is attached to each community to monitor the set of local and distributed agents so as to resolve the problem of distributed deadlock that could arise in any of the communities.

3 Case study/Experimental Set-up

Implementation of deadlock detection in VKC has been done under the Jade system, a Java based software development framework that conforms to FIPA standards for intelligent agents [JAD08]. The efficiency of JADE platform for agent development has been tested in a scenario where the number of agents and messages are increased, to test the efficiency of agent creation and scalability [C+04]. These researchers found out that JADE is a very efficient environment limited only by the limitations of the standard Java programming language. The environment does not introduce substantial overhead and JADE scales well when messages and agent movement are substantially increased. The fact that JADE conforms to all the appropriate FIPA standards and enables portable and easily maintainable agent development using the Java language, makes it a perfect framework in which to develop a multi agent-based system for deadlock detection and recovery in knowledge management using virtual knowledge communities. The implementation intends to address the problem of knowledge sharing activities of the agents, so all the other behaviors of the agents are not considered in the implementation.

3.1 Overview of the prototype

DeadlockDetectionCommunity: This class detects deadlock and resolves all kind of deadlock that could arise in any of the virtual knowledge community. It monitors every request of each agent in the communities. Comprise of Shadow agents and detector agents. The agents are created by the host community and are responsible for maintaining the resources used by a particular community agent, following it through the network and for initiating the deadlock detection phase. Furthermore, they assess the data gathered by detection agents to initiate deadlock resolution and detect and recover from faults during the deadlock detection process. They represent a portion of the global wait for graph. Shadow agents are responsible for recovering from failures during the deadlock detection phase.

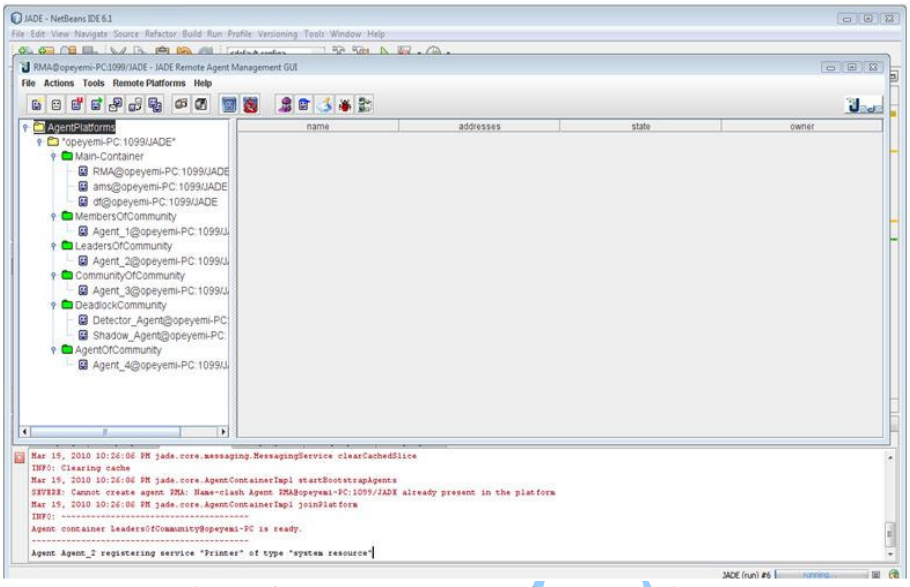

Figure 4. The Yellow page of agent services

Figure 5 depicts the welcome window where the user has to press the login button.

Figure 5. LoginWindow

Each member of the VKC must have the username and password in order to access the resources in the VKC.

After the login button has been pressed, an authentication window appears as shown in figure 6 where the user has to supply the username and the password.

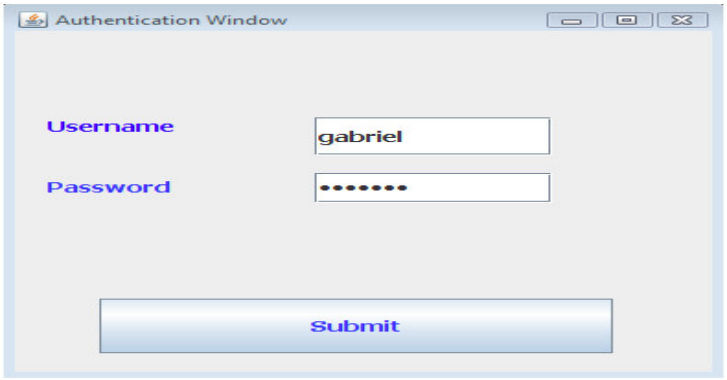

Figure 6. Authentication Window

When the username and password are entered, the agents are now free to request or share a resource in the VKC. Figure 7 shows the window where the user has to provide the agent's name, resource needed and the usage time that will be required to use the resource. After then, the resource is now granted.

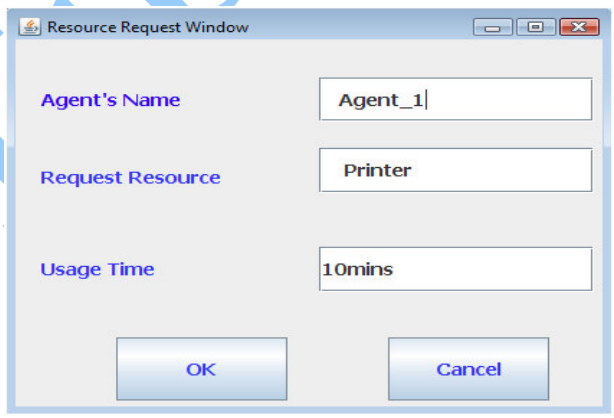

Figure 7. Request Resource Window

In the other hand, if the resource is in use, the shadow agent displays a message box indicating that the particular resource requested is in use by another agent as shown in figure 8. The system was tested on two different faculties knowledge network in a university environment.

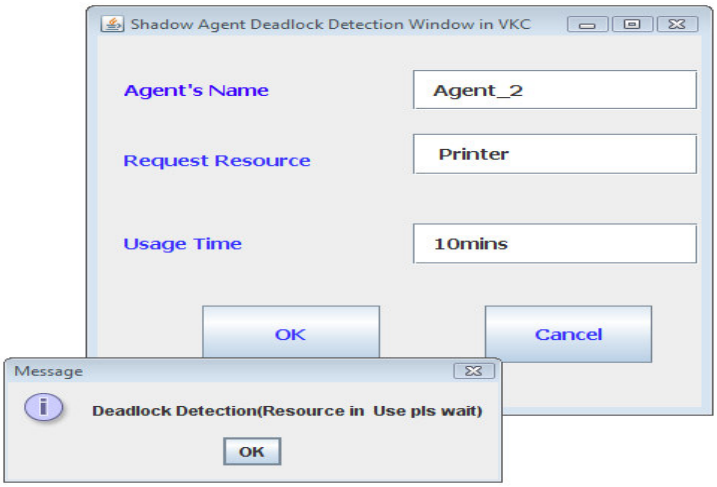

Figure 8. Shadow Agent Deadlock Detection in VKC

4 Results and Discussion

All agents used in our system as highlighted in figure 4 of this paper were created under the Java Agent Development Framework (JADE) system. The implementation addresses the problem of deadlock problem in knowledge sharing activities of the agents.

The agent based deadlock detection in virtual knowledge community approach enables agents to exchange knowledge in a formal manner with other agents on a peer-to-peer basis. As long as agents are in the same community, they have the potential to exchange knowledge, and since agents can join any community they wish to, they have the possibility to exchange knowledge with any agents in the entire organization in a formal manner, as long as they share a domain of interest, meanwhile taking care of any possible intrusion. Our model will enhance free knowledge sharing among the agents in VKC. In addition, our model will be of interest to most organizations as it will help in the free passage of knowledge from employer to employee or vice-versa.

Conclusion

In this paper, deadlock detection in virtual knowledge communities have been studied with the introduction of shadow and detector agents. A novel equation has been derived for deadlock detection in virtual knowledge communities. The paper has indicated the need to develop efficient systems capable of resolving deadlock hindrances during the sharing of knowledge in VKC. Future work should concentrate more on developing a more intelligent and robust system that will be able to detect and resolve deadlock in heterogeneous network communities.

References

[Ash00] **B. Ashfield** - *Distributed Deadlock Detection in Mobile Agent Systems*, M.C.S. Thesis, Carleton University, 2000.

[BM02] **M. Bonifacio, P. Maret** - *Knowledge Nodes: the Building Blocks of a Distributed Approach to Knowledge Management,* Journal of Universal Computer Science, Vol. 8 No. 6, pp. 652–661, 2002.

[C+04] **Krzysztof Chmiel, Dominik Tomiak, Maciej Gawinecki, Pawel Karczmarek, Michal Szymczak, Marcin Paprzycki** - *Testing the efficiency of the Jade platform*, Proceedings of the ISDP conference, Los Angeles, IEEE CS Press, pp. 49-56, 2004.

[Fer97] **J. Ferber** - *Les systems multi-agents: un aperu general*, Technique et Science Informatiques, Vol.16,No.8, 979-1012, 1997.

[HC07] **K. Huang, Y. Fang Chung** - *Efficient migration for mobile computing in distributed networks.* Elsevier, February 2007.

[HS99] **M. Huhn, L. Stephens** - *Multi-agent systems and societies of agents*, in Multi-agent Systems, A Modern Introduction to Distributed Artificial Intelligence, MIT Press, Cambridge, USA, pp. 79-120,1999.

[IM80] **S. Isloor, A. Marsland** - *The deadlock problem: An overview*. Computer (Sept.), 58-70, 1980.

[JAD08] **JADE,** Jena's development team, available at http://jena.sourceforge.net/index.html.(2008)

[J+04] **C. Jiannong, Z. Jingyang, Z. Weiwei, C. Daoxu, A. Jian** - *Mobile Agent Enabled Approach for Distributed Deadlock Detection,* Springer-Verlag Berlin Heidelberg, 2004.

[NVB08] **M. Nariman., G. Vahid, H. Behrouz -** *Monitoring Multi-Agent Systems for Deadlock Detection Based on UML Models,* Department of Electrical and Computer Engineering, Schulich School of Engineering, University of Calgary, Canada, 2008.

[OEC96] **OECD** - *The Knowledge Based Economy*, Paris, STI, pp.57, 1996.

[PJ09] **Maret Pierre, Calmet Jacques** -*Agent-Based Knowledge Communities*, International Journal of Computer Science and Applications Technomathematics Research Foundation Vol. 6, No. 2, pp 1-18, 2009.

[Sin89] **M. Singhal** - *Deadlock Detection in distributed Systems*, IEEE, 1989.

[Wii93] **K. M. Wiig** - *Knowledge Management Foundations*, Schema Press, Arlington (Texas), 474 p, 1993.