Journal of Mines, Metals and Fuels, 71(5): 671-677; 2023. DOI: 10.18311/jmmf/2023/34168

Print ISSN : 0022-2755

Journal of Mines, Metals and Fuels

Contents available at: www.informaticsjournals.com/index.php/jmmf

Maximizing the Efficiency of Multi Agents based Systems in a Synchronous Environment

Chandanita Thakur[1*], Shibakali Gupta[2]

[1]Department of Computer Science and Engineering, Swamy Vivekananda University, Barrackpore, West Bengal, India. Email: thakur.chandanita@gmail.com*

[2]Department of Computer Science and Engineering, UIT, Burdwan University, West Bengal, India.

Abstract

Evaluation of performance and efficiency is an important ingredient for any system design. A system can be comprised many components. Components can be hardware or software. Each component needs technical support to have proper cooperation among them. Cooperation can be better provided if they have autonomy, reasoning, proactivity, mobility and capability of learning. These qualities are inherent for Agents. Software Agents can cooperate, coordinate and negotiate much better than the way we cooperate, coordinate and negotiate with each other. So it is obvious that the system designed with the cooperation of multiple Agents i.e. Multi Agent System can produce higher throughput than most of the other existing systems .So the aim of the present work is to analyze and make a relationship between multiple Agent properties like proactivity, learning, reasoning, mobility, etc. and finally proposing an efficient algorithm to calculate the efficiency of Agents based on their properties and make them to work simultaneously in a multi Agent environment, for satisfying several goals.

Keywords: *Multi Agent System (MAS), Agent Properties, Efficiency, Proactivity.*

1.0 Introduction

Any system works well if its components work well. Components work well if they have proper cooperation[1]. Cooperation[2] comes from some inbuilt qualities[3]. The qualities can be learning, reasoning, mobility, proactivity[4] etc. These qualities are inherent for Agents[5]. So the maximum efficiency[6] can be expected from a system if it is designed[7,8] with the help of multiple Agents. Designing the architecture[9,10] of a multi Agent system is easy compared to make Agents function in an efficient and synchronous way. The term 'efficient' implies that the most efficient Agent should be selected from each goal. The term 'synchronous' implies that one Agent can participate for serving many goals. But if efficiency is combined with the term synchronous it means a lot! It means, the most efficient Agent should be chosen from each goal. As the same Agent can participate in several goals to achieve synchronization then the most efficient Agent called from one goal can be busy serving the purpose of other goal. So, first goal cannot get the Agent with highest efficiency until and unless that particular Agent is freed by the second goal. There are two solutions of this problem, either first goal has to wait for the second goal to release the Agent or to start the work with next efficient Agent. To find out the solution of such kind of problem which can be caused because efficiency maximization in Multi Agent based synchronized environment – this paper is designed.

The proposed work is started with analyzing the Agent properties such as autonomy, reasoning, proactivity, learning etc. and finding out the relationships and interdependence between the properties. Then the work is continued with an efficient algorithm to calculate the efficiency of Agents based on their properties, acting in a synchronized multi Agent based environment for serving several goals.

[*]Author for correspondence

2.0 Properties of Agents

All the Agents should exhibit some properties. Some properties of the agents are described below:

1. Autonomy: It is the property of the agents which helps them work independently and exhibit control over their internal state. If the agent is autonomous its behaviour will be determined by its own experience.
2. Learning: Learning allows agent to operate initially in an unknown environment. Learning modifies the performance of an Agent. It is required for true autonomy.
3. Reasoning: Reasoning enables agent to think something in a logical way in order to form a conclusion or judgement.
4. Proactivity: Proactivity enables agents to maintain an ongoing interaction with its environment and responds to challenges that occur in it. An agent may be able to exhibit goal-directed behaviour by taking the initiative.
5. Reactivity: Reactivity makes an Agent to perceive and act to some degree, on its close environment. A reactive agent can respond in a timely fashion to changes, that occur around it.
6. Rationality: For each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has.
7. Mobility: It is the ability of agent to move around the electronic circuit or its environment.

3.0 Why the Property Proactivity is Choosen?

From the thorough state of art review, (need to go through reference paper 13-24) we can find out that there is no work done on the relationship of agent properties. Some authors have utilized the properties of agent to achieve their goal, but they have selected that very property in their paper as a solution of the basic need they wanted to fulfill. The reason for choosing that very property is not justified in those papers. Those properties are learning, reactivity, autonomy, rigidity, reasoning, rationality, negotiation etc. One more important thing is, in recent papers there is no work done by utilizing proactive agent from 2012 to 2016.

From the discussion of Agent properties mentioned in this paper in section (II), it is clear that proactivity is an essential ingredient for an Agent, if maximum efficiency is needed to achieve from some real-time system where Agent is applied. In our current work our need is to maximize the efficiency. So in our recent paper, we took an attempt to calculate the efficiency in terms of Proactivity.

4.0 Relationship Between Agent's Properties

Along with proactivity, reasoning and learning etc. also are very important properties that the agent must have. So the Agents to be used for solving the above mentioned challenge, (introduced in Section II) efficiency can be calculated (in terms of Proactivity) by taking other properties as input.

We have gone through many books[25,26] and papers[13-24] to find out the dependencies and relationships between the agent properties, as very few works has been done on it. As a result of which, we have tried to concentrate on some important properties where, the change in one property can cause the change in other properties. Specially in our present work we have tried to find out the relationship of few among all properties which affects Proactivity, because our aim is to calculate efficiency in terms of Proactivity.

So the present section of this paper, first analyzes the relationship and interdependency between agent's properties and then calculates efficiency in terms of proactivity by taking reasoning, learning and mobility etc. as input.

$$f1: \text{Reasoning} \xrightarrow{\text{increase}} (\text{Rationality, Proactivity}) \quad \ldots (1)$$

[Let say, increase in Reasoning causes Rationality to increase a factor of x1 and increase in Reasoning causes Proactivity to increase a factor of y1. Value of x1 and y1 can be integer or float or exponential. Based on the values of x1 and y1, we can tell whether rationality and proactivity has linear or exponential relationship with reasoning]

$$f2: \text{Learning} \xrightarrow{\text{increase}} (\text{Reactivity, Autonomy}) \quad \ldots (2)$$

[Let say, increase in learning causes reactivity to increase a factor of x2 and increase in learning causes autonomy to increase a factor of y2. Value of x2 and y2 can be integer or float or exponential. Based on the values of x2 and y2, we can tell whether reactivity and autonomy has linear or exponential relationship with learning]

$$f3: \text{Autonomy} \xrightarrow{\text{increase}} (\text{Proactivity}) \quad (3)$$

[Let say increase in autonomy causes reactivity to increase a factor of y3. Value of y2 can be integer or float or exponential. Based on the value of y2 we can tell whether Proactivity has linear or exponential relationship with autonomy]

So we can write Proactivity = y1 * Reasoning [From 1]

Autonomy = y2 * Learning [From 2]

Proactivity = y3 * Autonomy [From 3]

From 2 and 3 we get

Proactivity = Y2 * y3 * Learning (4)

So some values can be assigned as input for learning, reasoning and mobility (other properties like autonomy, rationality etc. can be neglected and efficiency can be calculated with the help of reasoning, learning and mobility because reasoning and learning are interrelated with autonomy, rationality) for each proactive agent and the values of y1, y2, y3 (x1, x2 if needed) according to the nature of the system. Hence, we can calculate the value of proactivity as:

Proactivity = y1 * Reasoning ...1

Proactivity = Y2 * y3 * Learning ...4

So total Proactivity = (y1 * Reasoning) +

(Y2 * y3 * Learning) + Mobility

[Mobility can be given as input]

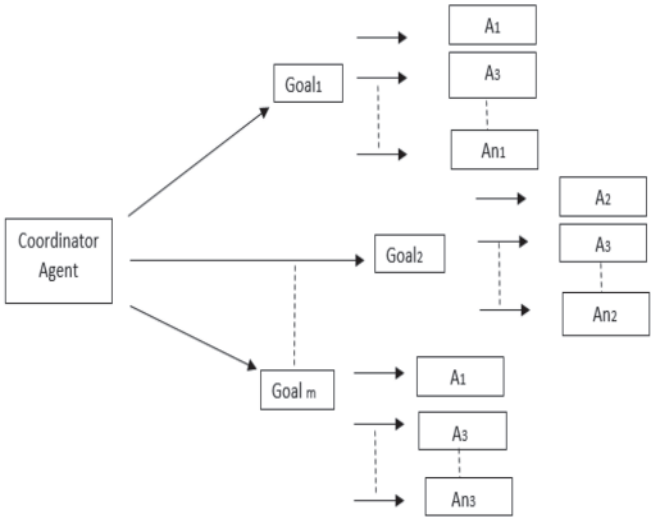

Figure 1: Agents interaction in a multi agent system

5.0 An Efficient Algorithm Based On Agent's Properties

A. Idea behind the algorithm

1. Situation (with real life example) – Let us consider an automated hospital implemented[27] by Multiagent system. Reception agent can be treated as the coordinator agent who can coordinate all the activities for a patient from admission to discharge and it will have many goals like searching for rooms, dealing with insurance agent, word agent and hospital management agent for patient admission, doing preliminary checkup by nurse agent, arrangement of doctor agent, investigation for patient by investigation agent, maintaining daily account information by account agent for individual patients etc. These goals can be executed in any order according to their priority and can be repeated any number of times according to the condition of the patient. Example, for emergency patient, after admission, immediately doctor will be called. Many agents can be involved in each goals, some of them can be common for many goals. As there can be some agents participating for many goals, there will be a situation (synchronization problem) where the same agent is selected by many goals. For example, same nurse agent can be engaged (say, by goal 3) in Patient's Investigation and at the same time can be called for making discharge summary for patient discharge (say, by goal 6)
2. Main Idea – In most of the multi agent system (based on the above situation), one main agent will coordinate the activity of all other agents. Main agent is named here as coordinator agent (CA). Coordinator agent is the starting agent which has many goals (let say m numbers of goals) to serve. According to the situation, coordinator agent can change the priority of the goal. For each goal there will be some specific agents. One agent can participate in multiple goals. External priority (decreasing number implies increasing priority) is set for all the goals and goals are initially arranged according to their priority. Let say goal1 is with highest and goalm is with lowest priority. So goal1 must be called by the coordinator agent when system will start executing. During the system execution, goal with any priority can be called according to the requirement of the system. For satisfying each goal, any number of Agents can be involved. Number of agents may vary from goal one to goal n (n can be any finite number).

Number of Agents are specific for a particular goal as shown in Fig.1. For goal1, required number of agents are 1 to n1. For goal 2, required number of agents are 2 to n2. For goaln, required number of agents are 1 to n3 etc. For efficiency maximization, system will try to call the most efficient agent from goal1. After that, goals and agents can be chosen in the following way:

1. After goal1 starts, goal 2 can also start (on or before completion of goal 1) with the most efficient Agent. The same process can be continued sequentially up to a certain goal (In the previous Agent based hospital example let say, goal1 is patient admission, goal 2 is Room allotment. So at the time patient is busy in taking admission, Room can be allotted and doctor can be called (may be goal 4).
2. If Coordinator Agent needs to repeat some goals according to system requirement, the sequential execution of goal will be interrupted and Coordinator will move the control back towards the increasing priority (decreasing priority value). If we consider the Automated Hospital Example, Investigation can be repeated again and again, Doctor can be called again and again etc.

3. As the same Agent can be involved in two or more goals (example, same person or Agent can take the patient during admission as well as during investigation) and same Agent can be the highest efficient Agent for more than two goals (later it can happen between two or more medium priority Agents also, which is shown in algorithm), so the same Agent can be called from more than one goals with a small fraction of time gap.
4. When one Agent is busy serving one goal (say goal1), and called from another (can be many others also) goal (say goal3), the execution of second goal (goal3) will be suspended. To solve that problem Coordinator can go for either of these three options
   i. Second goal can wait for the first goal to release that particular Agent up to a time slice t
   ii. Second goal can start executing with the next efficient Agent, belongs to that goal.
   iii. If the next efficient Agent is also busy next to next efficient Agent should be called.

   Until and unless the next efficient Agent is free, the same process will be continued recursively.
5. When one Agent is busy, the busy state of the agent can be intimated to all the agents . In the same way whenever a busy agent is getting free, (shown in algorithm) can be informed to all the goals. This method can help Coordinator to take decision about which goal to start with what agent.

B. Introduction of matrices used in algorithm

For a specific system (here the system is multi agent based), number and nature of goals (different types of works can be done by the system) must be same for all the applications. But out of all, how many goals are needed and the priority of the goals can change from application to application where the system is going to be used.

Step 1: Before starting any application, priority should be assigned to all the goals and will be intimated to Coordinator agent. In the present work it is assumed that the goals in Fig.1 is arranged according to descending priority (increasing priority means decreasing priority value).

Step 2: Next the system will take the value of Agent [m][n] matrix which gives the details of agent's participation in each and every group. Consider the 1st row of agent matrix given in Fig.2.

For goal 1, agent 1, agent 3, agent 4 etc. are involved, so the value 1, 3, 4 are entered in columns 1, 3, 4. As agent 2 and agent n is not participating for goal 1, so the value 0 is entered in column 2 and n etc.

Step 3: Next step is to calculate the efficiency for each Agent having non zero value by using calculate_efficiency function. Efficiency values for all the agents are stored in

Agent[m][n] =	A1	A2	A3	A4	An
G1	1	0	3	4	... 0
G2	0	2	3	0	... n
G3	1	0	0	4	... 0
G4	0	0	3	4	... 0
.					
Gn	1	0	3	4	... 0

Figure 2: Input agent matrix

efficiency [m][n] matrix. (if a particular agent is not present for a specific goal, efficiency of that agent is taken as 0). For example in automated hospital, nurse agent is not responsible for patient admission (say goal1) but responsible for calling doctor (say goal 3), investigation (say goal 4) etc. If nurse agent is named as A2, for goal 1 its entry will be 0.

Efficiency [m][n] =	A1	A2	A3	A4	...An
G1	.5	0	.9	.8	...0
G2	.8	.1	.3	0	6
G3	.7	0	0	.4	...0
G4	0	0	.3	.4	...0
.					
Gn	.2	.2	3	.9	...0

Figure 3: Efficiency matrix

Step 4: Sort_efficiency [m][n] matrix will take the copy of the values of efficiency matrix. Then the values will be sorted using swap function in descending order and will be stored. Agent with highest efficiency will be called for each goal.

Sort_Efficiency [m][n] =					
G1	.9	.8	.5	0	...0
G2	.8	.6	.3	.1	...0
G3	.7	.4	0	0	...0
G4	.4	.3	0	0	...0
.					
Gn	.9	.3	.2	.2	...0

Figure 4: Efficiency matrix sorted in descending order

The same agent can be involved in more than one goals. Ex. If agent 1 is serving for goal1 and at the same time called from goaln., goaln cannot be served. So after a finite time(t)

expires goaln can be started with the agent with next higher efficiency. If that is also busy, agent with next higher efficiency can be called. The intension of making sort_efficiency[m][n] matrix is to provide the next efficiency value when the agent, holding the present efficiency (treated as highest efficient presently) value, is busy to serve some other goal.

Drawback of sort_efficiency [m][n] is, it does not give the agent number with highest efficiency. For all the goal highest efficient agent is the agent present in the first column, but for goal 1 it is agent 3, goal 2 it is agent 1, goal 3 it is agent 1, goal 4 it is agent 4, goal n it is agent 4 etc. So from the highest efficiency value, it is not possible to predict the agent number from the present matrix. In efficiency matrix, efficiency is sorted here, not agents. But together from sort Efficiency [m][n] and efficiency[m][n] matrix it is very clear that 9 is the efficiency value of agent 3. (As the agents are not sorted, so first to last column, cannot be marked as A1 to An in Fig 4 as it is marked in Fig.3). So, to find out the agent number corresponding to a particular efficiency, some mapping is needed between the two matrices sort_efficiency [m][n] and efficiency [m][n].

Step 5: So next step is to built a matrix highest_ efficiency where only first element (first column) will have the highest efficiency value for the corresponding goal. To save the original column index of that highest efficiency value (which is nothing but the original column index of the Agent having highest efficiency) another matrix Index [m] is needed.

Highest_Efficiency [m][n] =

G1	.9 - - - ... -
G2	.8 - - - ... -
G3	.7 - - - ... -
G4	.4 - - - ... -
.	
Gn	.9 - - - ... -

Figure 5: Matrix having highest efficiency value for each goal

The above concept can be explained with the help of highest_efficiency array. Initially highest_efficiency matrix will take the copy of values of efficiency matrix. Consider any one row, let us take last row efficiency value, which are .2 .2 .3 .9 …0 . If some sorting met

Hod (explained in algorithm) is applied in descending order only for first iteration (i.e for j=1 and k=1 to n-1 shown in algorithm), with calling of swap() function when the former value is greater than later. First swap function will be called For j=1 and k=3 and n^{th} row will be sorted as .3 .2 .2 .9 …0 (because .3>.2). Second swap() function will be called for j=1 and k= 4 n^{th} row will be re-sorted as .9 .2 .2 .3 …0 (because .9 > .3). For the present situation this is the sorted n^{th} row where k gives the original column index of Agents (present in Agent or Efficiency matrix in Fig.2 and Fig.3). So that 'k' value should be stored and updated (last updated value will be the column index of the original Agent with highest efficiency) in an array every time when the swap() function is called. When swap function will be called for the last time, that final k value will be stored in Index matrix and will provide the column value of highest efficiency matrix (calculation of row value is not dependent on matrix, shown in algorithm).

```
Index[I][J] = [3 2 1 3 3]
Algorithm
Define maximum_no_of_group m
Define no_of_ maximum_ agents_ in_a_group n
Float reasoning [m][n];
Float learning [m][n];
Float mobility [m][n];
Read y1, y2, y3;
G [m];
Efficiency [m][n];
Sort_ Efficiency [m][n];
Highest_ Efficiency [m][n];
Agent [m][n];
Read Agent [m][n] //Read the value of agents
for a particular goal as
 (i) 1 in first column, if
     Agent 1 is needed
(ii) 2 in second column, if
     Agent 2 is needed
     ...
(iii) 0 if that particular Agent
     is not needed
 /* Coordinator Agent (CA) will start with first goal
 from m number of assigned Goals */
 /*Goals are arranged according to priority i.e goal1
 is having highest priority*/
 For I ← 1 to m
    Select Goal G[I] from CA
        For J ← 1 to n
            Select Agent[I][J] for Goal G[I] //Select
                    first Agent from first Goal
 if Agent [I][J] = 0
    Efficiency [I][J] = 0
 else
Efficiency [I][J] = Calculate_Efficiency (Agent[I][J])/*
Efficiency matrix will hold the values of efficiency For all the
Agents for all the goals from 1 to m*/
 For I ← 1 to m
 For J ← 1 to n
 Sort_Efficiency [I][J] = Efficiency[I][J]
      //Copy the value of Efficiency
      to Sort_Efficiency
```

```
 Highest_Efficiency [I][J] = Efficiency [I][J]
 //Copy the value of Efficiency
to Highest_Efficiency
For I ← 1 to m //Sort the efficiency of
                agents for each goal
For J ← 1 to n
   For K ← J+1 to n-1
     If Sort_Efficiency [I][J] < Sort_Efficiency [I][k]
        Swap (& I, & J, & k)
```

/* For each group, Agents will be sorted in descending Order. So first Agent will be the agent with highest efficiency*/

```
For I ← 1 to m //Sort the efficiency of
          agents for each goal
  For J ← 1
    For K ← J+1 to n-1
      If Sort_Efficiency [I][J] < Sort_Efficiency [I][k]
         Swap (& I, & J, & k)
 x=J    //J is the index of highest efficiency
         agent in each goal
 Index [I] = x // Index [I] will store the index
 of highest efficiency Agent in each goal
```

/* Highest_efficiency is a matrix having m rows and n columns where first column stores the most efficient Agent from each goal */

/* Index array holds the column index of Agent with highest efficiency*/

/* So the Agent with highest efficiency will be called first */

Coordinator agent can execute wait (goal no) system call when it is not necessary to start and can make signal (goal no) = true, when want to start.

```
I ← 1
While (true)     // Implies Agent number
                  is in between 1 to m
If signal (I)= = true
q = index[I]     //q- is column index of highest
                 efficiency agent for goal 1
No = 2     // If first Agent is not free, need
           to call second efficient agent
from Sort_Efficiency [I] [No]

If Agent [I] [q] is free allocate Agent[I][q]
else
Number = Calculate_next _efficient
            _Agent (Sort_Efficiency[I][No])
Allocate Agent [I][Number]
else
i++        //go for next goal & check
           for semaphore value
```

/* The above mentioned procedure for calling next efficient agent can be implemented with the help of synchronization */

/* One Agent may participate to fulfil several Goals and can be called at the same time .If a finite time(t) expires, the Agent with next higher efficiency can be called */

```
If Agent [I][J] is the Agent with highest efficiency
```

/* Check Agent[I][J] for concurrent use*/

/* use1 and use 2 are two synchronized function. If an Agent is busy serving one goal (say goal 2) and called from other goal (say goal 4), it (goal 4) has to wait .If a finite time (t) expires, CA will start the goal with next efficient agent by calling calculate_next_efficient_agent (sort_efficiency [I][J], I) */

```
Check_concurrent_use (Agent[Ii][Jj])
For J ← Jj
For I ← 1i
For k ← 1 to m
If signal (Agent [I][J]) = = signal (Agent [K][J])
   Use1 (Agent [I][J])
      else
         use 2 (Agent [I][J])
boolean valueset=false
synchronized float use1 (Agent[I][J])
if (! Valueset)
   try
     wait();
        Catch (Interrupted Exception e)
           Print (Agent [I][J]can not be used)
 If wait (Agent[I][J]) > t
Calculate_next_efficient_Agent_Sort_Efficiency
        ([I][No], I)
notify all(); //Will notify other goal about the
         status of present Agent
valueset = false
Notify();
Return (Proactivity (Agent [I][J]))
synchronized float use2 (Agent [I+1][J])
  if(valueset)
     try
        wait();
Catch (InterruptedException e)
        Print(Agent[I+1][J]can be used)
valueset = true
Notify All ();
Calculate_Efficiency (Agent [x][y])
Proactivity = (y1 * Reasoning) + (Y2 * y3 * Learning)
+ Mobility[x][y]
        Return (Proactivity)
int Calculate_next_efficient_Agent (Sort_Efficiency[p]
[No], p)
For I ← p
  For J ← 1 to n
If Efficiency [I][J] = = Sort_Efficiency [p][No]
Exit ();
If Agent [I][J] is free
Return J;
```

else

Calculate_next_efficient_Agent (Sort_Efficiency[p][No+1])

/* Recursive function Calculate_next_efficient_Agent will be keep on calling the next Agent if the present highest efficiency Agent is busy and will send the column index of available highest efficiency Agent */

void Swap (int *I, int *J, int *k)

Temp = Sort_Efficiency[I]

Sort_Efficiency[I][J] = Sort_Efficiency[I][k]

Sort_ Efficiency[I][k] = Temp

Result : For the same work, goals (In turn efficiency of agent) can be selected in different ways. The different ways can be called as choice (C1, C2, C3).This is the graph between choice and efficiency. Choice 5 i.e C5 is best.

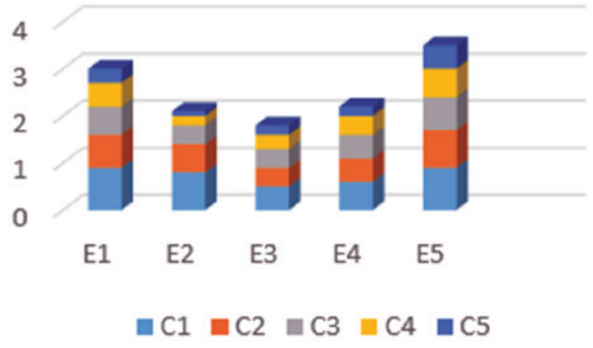

Figure 6: Graph between Choice and efficiency

6.0 Conclusions

For any system, efficiency, performance and throughput are very closely related. If efficiency is increased, performance will obviously increase; and increase in performance will in turn increase the throughput of the system. Maximizing throughput should be the intention of designing a system and designing of efficient algorithm is one of the most important way to make the system components to perform in coordinated way to maximize the throughput. This paper is presented a novel algorithm which helps any Multi Agent[11,12] based system to perform with highest efficiency in a synchronized way. The former part of the paper described the relationship of Agent's efficiency with its properties and the later part showed how efficiently Agent can be utilized to maximize the performance in a synchronized environment. The later part of the paper has been experimented and the result has been included in the paper. This work can be implemented by using JADE to fulfil the need of efficiency maximization raised in Multi Agent System.

7.0 Reference

1. Michael Wooldridge, 'An Introduction to Multi Agent Systems', Department of Computer Science, University of Liverpool, UK, JOHN WILEY & SONS, LTD.
2. Katia P.Sycara, 'Multi Agent Systems', 1998.
3. M. E. Bratman, 'Intentions, Plans, and Practical Reason', Harvard University Press, Cambridge, MA, 1987.
4. Kresimir Jurasovic Gordan Jezic Mario Kusek, 'A Performance Analysis of Multi-Agent System'.
5. Amit K . Chopra and Munindar P. Singh, 'An Architecture for Multi-Agent Systems : An approach Based on Commitments', Universit'a degli Studi di Trento, North Carolina State University.
6. Winikoff. M, 'Implementing commitment-based interactions', Proceedings of the 6th International Joint Conference on Autonomous Agents and Multi Agent Systems 2007.
7. A. Garcia et al, 'Separation of Concerns in Multi-agent Systems : An Empirical Study', In: Software Engineering for Multi-Agent Systems II, Springer, LNCS 2940, April 2004.
8. A. Pace et al., 'Assisting the Development of aspect-based MAS using the Smart Weaver Approach', In: "Software Engineering for Large-Scale MASs", LNCS 2603, March 2003.
9. K. Potiron et al., 'Multi-Agent System Properties', Chapter 2, From Fault Classification to Fault Tolerance for Multi-Agent System, Springer Briefs in Computer Science, DOI: 10.1007/978-1-4471-5046-6_2, _ The Author(s) 2013.
10. Sara Maalal, Malika Addou, ' A new approach of designing Multi-Agent Systems With a practical sample ', IJACSA International Journal of Advanced Computer Science and Applications, Vol. 2, No. 11, 2011.
11. Mihaela Oprea, 'Applications of multi-agent systems', University of Ploiesti, Department of Informatics, Bd. Bucuresti Nr. 39, Ploiesti, Romania.
12. Onn Shehory, 'Architectural Properties of Multi-Agent Systems', CMU-RI-TR-98-28, The Robotics Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, December 1998.
13. Elaine Rich, Kevin Knight, 'Artificial Intelligence' isbn=0071008942.
14. Kevin Knight, 'Artificial Intelligence' isbn=0070522634.
15. Chandanita Thakur, Dr. Shibakali Gupta Agent oriented Intelligent Treatment Coordination System.