Task Petri Nets for Agent based computing

PRAJNA DEVI UPADHYAY[1] SUDIPTA ACHARYA[2] ANIMESH DUTTA[3]

[1] Indian Institute of Technology, Delhi,
Department of Computer Science and Engineering,
New Delhi-110016, India
kirtu26@gmail.com

[2] Indian Institute of Technology, Patna,
Department of Computer Science and Engineering,
Patna-800005, India
sudiptaacharya.2012@gmail.com

[3] National Institute of Technology,
Department of Information Technology,
Durgapur-713209, India
animeshrec@gmail.com

Abstract. Agent Based Computing is an emerging area of research for the last couple of decades. The agents must be capable of maintaining the inherent dependencies between different tasks or handling resource constraints while working together. There is a need of a formal tool that can represent this autonomous behavior and task delegation property of agents. This paper proposes a new formal tool called Task Petri Nets to represent the collaboration of agents in Multi agent system (MAS). A Task Petri Nets is an extended Petri Nets tool which can represent the autonomous behavior of agents while executing any task maintaining the happened before relationships and handling resource constraints. It can also evaluate the performance of the system using three metrics- Total Execution Time, Agent Utilization and Resource Utilization.

Keywords: Agent; Multi Agent System; Petri Nets; Task Assignment; Resource Constraint.

(Received February 23rd, 2013December 8th, 2010 / Accepted July 30th, 2013July 16th, 2011)

1 Introduction

The history of computing has been marked by various ongoing trends [25]. Computing has become ubiquitous due to availability and interconnection of large number of processors. The aim of many theoretical works has been to portray computing as a process of interaction between humans and machines. The complexity of tasks that we are capable of automating and delegating to computers has grown steadily. We are moving away from machine-oriented views of programming to ideas that more closely reflect the way we understand the world. The advancement from assembly level programming to procedures and functions and finally to objects has taken place to model computing in a way we interpret the world. But there are some inherent limitations in an object which makes it incapable of modeling a real world entity. Therefore we move to agents, which models the real world problem in a better way.

An agent is a computer system or a software entity which can act autonomously in an environment. Agent autonomy relates to an agent's ability to make its own decisions about what activities to do, when to do, what type of information should be communicated and to whom, and how to assimilate the information received. Thus, an intelligent agent inhabits an environment and is capable of conducting autonomous actions in order to satisfy its design objective [14], [18], [19], [24], [25]. The environment is the aggregate of surrounding things, conditions, or influences with which the agent is interacting. Information is percepted by the agent and so this information is also called 'percepts'. The agent works on the percepts in some way and produces 'actions' that affect the environment.

Multi agent systems are computational systems in which two or more agents interact or work together to

perform a set of tasks or to satisfy a set of goals [15], [23], [25]. Agents of a multi agent system need to interact with each other to fulfill their common or individual objectives. A multi-agent system can be studied as a computer system which is concurrent, asynchronous, stochastic and distributed. A multi agent system allows coordinating the behavior of agents which are interacting and communicating in an environment. It allows the decomposition of complex task into simple sub-tasks so that the problem can be addressed at a more granular level.

Petri Nets and Color Petri Nets [5] are graphical tools for the formal description of systems whose dynamics are characterized by concurrency, synchronization, and mutual exclusion, which are typical features of distributed environment. A Basic Petri Net structure is a tuple, $PN = (P, T, I, O, TOK)$ where

- $P = \{p_1, p_2, \ldots, p_x\}$, where x$\geq$ 1, is a finite set of places.
- $T = \{t_1, t_2, \ldots, t_y\}$, where y$\geq$ 1, is a finite set of transitions, $P \cap T = \emptyset$ i.e., the set of places and transitions are disjoint.
- I: $P \rightarrow T$ is the Input Arc, a mapping from places to bags of transitions.
- O: $T \rightarrow P$ is the Output Arc, a mapping from transitions to bags of places.
- $TOK = \{TOK_1, TOK_2, \ldots, TOK_z\}$, where z$\geq$ 1 is a finite set of dynamic markings on places.

Our work proposes a new formal tool called Task Petri Nets which represents the autonomous behavior of the agents in a Multi Agent System. The MAS consists of a number of agents with different capabilities and according to the requirement, each task should be assigned to some capable agent. Initially, a user query is submitted to the MAS. This query is mapped to a task by an interface agent. The interface agent also decomposes this task into a number of sub-tasks and determines the happened before relationships and resource requirements of the sub-tasks. But, all the sub-tasks cannot be executed concurrently because of the existence of some inherent dependencies and some resource constraints. Thus, the paper addresses the problem of task assignment with a set of constraints and suggests a formal solution for the same in the form of Task Petri Nets. The Task Petri Nets model helps to analyze the performance of the system with the help of three metrics. A case study has been modeled with Task Petri Nets, the scenario has been simulated and measured performance with the help of MATLAB tool.

2 Related Work

There has been a wide spectrum of research work in the area of Multi Agent Based Computing. Most of these are concerned with the task assignment problems, determining the assignment of tasks to agents keeping in mind the constraints associated with them. In [27] authors consider a problem where only a specific number of agents can perform a certain task. This means the problem is agnostic to the actual coalition serving a given task and that their algorithm considers a much smaller search space. Also, they assume a central planner that allocates tasks to agents i.e they follow a centralized method. Similarly, in [4] authors use a centralized mechanism based on random neural networks to allocate responders to perform a number of rescue tasks. The drawback of using centralized method in paper [27] and [4] is overcomed in [10], where authors consider completely decentralized solutions to an allocation problem which considers teams of agents but ignores the spatial constraints and show how different DCOP formulations of the problem result in different degrees of computational and communication efficiency when used with typical DCOP algorithms such as ADOPT [12] or DPOP [16]. Authors of paper [20] and [3] have applied DCOP and other decentralized heuristics to the general assignment problem (GAP). Now, while these approaches consider heterogeneous agents (i.e., agents with different capabilities) and execution constraints for tasks (e.g., two tasks that must be executed at the same time), they ignore the benefit of forming coalitions of agents (i.e., with synergistic capabilities) to work on the same task. Problems of neglecting spatial constraints in [10] and benefit of forming coalitions of agents in [20] and [3] are overcomed in [17], where authors model the RoboCupRescue domain in terms of a Coalition Formation with Spatial and Temporal constraints. Here a set of agents, such as rescue agents in search and rescue, must work together to perform a set of tasks, often within a set amount of time. Authors provide a Distributed Constraint Optimization Problem formulation of the problem and show how to solve it using the Max-Sum algorithm. Based on this, they develop the novel F-Max-Sum algorithm that improves upon Max-Sum in order to deal with disruptions in its underlying factor graph more effectively. This paper lacks of expressing the fact that how task allocation can be done such in a way where agent will complete all tasks in minimum time which is shown in [8]. In this paper authors propose a distributed algorithm to get efficient distribution of tasks across heterogeneous agents. Each task has an execution time which is different if different agents execute them. Here it is

assumed that run time to perform a task by an agent is known initially. Authors have shown how to assign tasks to agents to finish all tasks in minimum time. But it is not realistic to know the execution time taken by different agents in different environments to complete a task before their actual execution. Also, the inherent dependencies between the tasks are not considered which may not allow the tasks to be executed concurrently. The paper lacks to evaluate the performance of the system. Methodology described in [7] presents scheduling algorithms on unrelated parallel machines. A novel distributed algorithm for multi agent task allocation problems is proposed in [9] where the sets of tasks and agents constantly change over time. But both in paper [7] and [9] the inherent dependencies between tasks are not considered which is considered in paper [21]. Here authors have considered dependencies between tasks and defined a dynamic ontology for coordination among agents in the MAS. They have defined different types of dependencies between tasks based on happened before relationship and resource constraints. But, the task delegation and autonomous properties of agents are not shown explicitly to solve the problem.

There are many forms of Petri Nets proposed over the years which serve specific purposes of a Multi Agent System. A Color Petri Net is a high level Petri Net which provides a graphical oriented language for design, specification, simulation and verification of systems. It is in particular well-suited for systems consisting of a number of processes which communicate and synchronize. It is a combination of Petri Nets and programming language where Petri Nets control the structures, synchronization, communication while functional programming language describes the resource sharing and data manipulation. It allows the definition of different data types for the tokens describing data manipulation, and for creating compact and parameterizable models. Petri Nets and Color Petri Nets have been widely used to describe the Multi Agent Systems for a long time. Color Petri Nets have been used in [1] to achieve agent scheduling in open dynamic environments. The representation of composite behaviors through Color Petri Nets have been done in [6]. In [2] Petri Nets have been used to model the abstract architecture for intelligent agents and structural analysis of the net provides an assessment of the interaction properties of Multi Agent Systems. Deadlock Avoidance in Multi Agent System is considered and is evaluated using the liveness and boundedness property of the Petri Net Model. Color Petri Net model is introduced to represent flexible agent interactions in [1] Agent Petri Nets [11] provides more importance to the internal state and behaviour of an agent in a Multi Agent System. Here, each token of a place represents an agent and the transition is capable of a set of functions that describes, in particular, the condition of its firings and relations between the agents. There are Predicate Transition Nets [26], [13] which are a high level formalism of Petri Nets for modeling and analyzing Multi Agent behaviors. In Multi Agent Systems, plans are built to specify how a set of agents accomplish their individual or common goal. Predicate Transition Nets allow us to make sure that the plans are reliable.

3 Scope of Work

The work done so far is concerned with task assignment problems i.e. assigning a set of tasks to a set of agents in MAS considering the constraints associated with each agent. But few authors have considered the dependencies between the tasks and resource constraints together for task assignment in MAS. Petri Nets are chosen to model this scenario because they are best suited to model the features of a distributed system. But, the existing Petri Net tools are incapable of doing so, so there is a need to extend its features. In this paper, we propose Task Graph to represent the dependencies between the tasks and Task Petri Nets to model the task assignment problem. A Task Petri Nets is an extended Petri Nets capable of representing agent autonomy and task delegation property. It can also evaluate the performance of the system using some metrics whose values can be obtained from the tool itself.

4 System Model

A Multi Agent System can be formally stated set of tuples $MAS = (S, t, A, inst_A, Q, T, G, R, inst_R, gen_task, gen_goal, req_res, able)$ where each tuples are described in detail below,

- $S = \{s_i \mid 1 \leq i \leq u\}$, where s_i is a state of the system. Therefore S is a set of states of environment.
- $t = \{t_1, t_2, \ldots, t_n\}$, each t_i, $1 \leq i \leq n$, is an atomic task which the agents in the system are capable of performing.
- $A = \{a_1, a_2, \ldots, a_p\}$, be the set of agents in the system. Each a_i, $1 \leq i \leq p$, is a type of agent in the system.
- $inst_A$ is a function defined as: $inst_A$: $A \rightarrow N$, A = set of agents, N = set of natural numbers $inst_A$ defines the number of instances of each agent available in the system.

- $Q = \{Q_1, Q_2, \ldots, Q_m\}$, where each Q_i, $1 \leq i \leq m$, is a query which the user may submit to the system.
- $T = \{T_1, T_2, \ldots, T_m\}$, where each T_i, $1 \leq i \leq m$, is a task, which is generated from a specific user query, $T_i \in \varphi(t)$
- $G = \{g_1, g_2, \ldots, g_m\}$, where each g_i, $1 \leq i \leq m$, is a goal which is generated for each task requirement in the system. A goal is some set of states achieved after successful completion of T_i. $g_i \in \rho$ (S)
- $R = \{r_1, r_2, \ldots, r_q\}$, where each r_i, $1 \leq i \leq q$, is a type of resource in the system.
- $inst_R$ is a function defined as: $inst_R : R \rightarrow N$, R = set of resources, N = set of natural numbers. $inst_R$ defines the number of instances of each resource type available in the system.
- gen_task is a function which maps each query to a task and is defined as: $gen_task : Q \rightarrow T$
- gen_goal is a function which maps each task to a goal and is defined as: $gen_goal : T \rightarrow G$
- req_res is a function which maps each task $T_i \in T$ to a subset of (RXI), where I is the set of non-negative integers.

  $req_res : T \rightarrow \rho\ (RXI)$, each task is mapped to a set of tuples indicating the resource and its number of requirement.
- $able$ is a function which maps each agent $a_i \in A$ to a subset of t, So $able : A \rightarrow \varphi(t)$

5 Task Graph

A Task Graph is a formal representation of the happened-before relationships between tasks. The happened before relationship is described in the following sub-section.

5.1 Lamport Happened Before Relationship

If a and b are two events taking place in a system, the expression $a \rightarrow b$ is read as 'a happened before b', which means all processes agree that before occuring event b, event a should occur [22]. The happened-before relation can be observed directly in two situations:

- If events a and b occur on the same process and the occurrence of event a preceded the occurrence of event b then $a \rightarrow b$ = TRUE
- If a is the event of sending a message m in a process and b is the event of receipt of the same message m by another process then $a \rightarrow b$ is also true. A message cannot be received before it is sent or even at the same time it is sent, since it takes a finite, nonzero amount of time to arrive.
- Happened-before is a transitive relation i.e. If $a \rightarrow b$ and $b \rightarrow c$ then $a \rightarrow c$.

5.2 Formal Definition of a Task Graph

A task graph G_T=(V_t, E_t) is an ordered pair consisting of:

- A set V_t=$\{V_{t1}, \ldots, V_{ta}\}$, where each V_{ti}, $1 \leq i \leq a$, is a vertex which represents a task $t_i \in T$ which should be performed.
- A set E_t=$\{E_1, \ldots, E_b\}$, where each E_i, $1 \leq i \leq b$, is an edge identified with an ordered pair of vertices (V_{tm}, V_{tn}), which represents the happened-before relationship between two tasks identified by those vertices.

Given a set of happened before relationships between tasks, we can form the task graph by listing all the vertices representing the tasks and drawing a directed edge from V_{ti} to V_{tj} if the happened before relationship $t_i \rightarrow t_j$ exists in the set of happened before relationships. Given two tasks t_i and t_j represented by the vertices V_{ti} and V_{tj} respectively in the task graph, we can infer the following about them:

- If there exists a directed path from vertex V_{ti} to V_{tj} in G_T, i.e. there exists a sequence of vertices and edges starting from V_{ti} and ending at V_{ti}, we can say $t_i \rightarrow t_j$.
- Two tasks t_i and t_j are concurrent if there is no directed path either from V_{ti} to V_{tj} or from V_{tj} to V_{ti}.
- There are no circuits and self loops in a task graph.

6 Petri Nets model of the problem

The existing Petri Net tool lacks for modeling this scenario. So, a new tool called Task Petri Nets is proposed which is described in following sections,

6.1 Task Petri Nets

Task Petri Nets is an extended Petri Net tool which can model a problem. It can be defined by set of tuples, $TPN = (P, TR, I, O, TOK, F_n)$

- $P = P$ is a finite set of places. It can be defined by union of 8 number of places. So we can write $P = (P_t \cup P_h \cup P_c \cup P_e \cup P_f \cup P_r \cup P_a \cup P_d)$. Places P_h, P_c, P_e, $P_f \in P$ exist for each task already identified by the interface agent. The description of the different types of places are follows:
  - P_h: Here the presence of a token indicates that the task represented by this place can run, i.e. all predecessor tasks that are required to be completed before this task are completed.
  - P_c: Here the presence of a token indicates that an agent has been assigned for the task represented by the token of corrosponding place P_h.
  - P_e: Here the presence of a token indicates agent and resources have been allocated for the task represented by the token of corrosponding place P_h and the task is under execution by the allocated agent.
  - P_f: Here the presence of a token indicates that the task represented by the token of corrosponding place P_h has finished it's execution.
  - P_r: This place exists for each type of a resource in the system. i.e. $\forall\ r_i \ \exists\ P_{ri}$ where, $r_i \in$ R and $1 \leq i \leq q$.
  - P_a: This place exists for each instance of an agent in the system. i.e. $\forall\ a_i \ \exists\ P_{ai}$ where $a_i \in$ R and $1 \leq i \leq p$.
  - P_t: Here the presence of a token represents initially identified task by the interface agent .
  - P_d: This place is created dynamically after the agent has been assigned for the task(in place P_c) and the agent decides to divide the tasks into subtasks. For each subtask, a new place is created.
- TR is the set of 5 transitions which can be represented as, $TR = (t_h \cup t_c \cup t_e \cup t_f \cup t_d)$. where t_h , t_e , t_f exist for every task identified by the interface agent. The description of different types of transition is as below,
  - t_h: This transition fires if a task represented by the token of place P_t is enabled i.e. all the preceeding tasks which should be completed for the task to start are completed.
  - t_c: This transition fires if the task represented by the token of corrosponding place P_h is assigned with an agent which is capable of performing it.
  - t_e: This transition fires if all the resources required by the task represented by the token of corrosponding place P_c are allocated to it.
  - t_f: This transition fires if the task represented by the token of corrosponding place P_e is completed.
  - t_d: This transition is dynamically created when the agent assigned for the task represented by the token of corrosponding place P_d decides to split the task further into subtasks. The subnet that is formed dynamically consists of places and transitions all of which are categorized as P_d or t_d respectively.
- I is the set of input arcs, which are of the following types,
  - I_1=P_t X t_h : task checked for dependency.
  - I_2=P_r X t_e : request for resources.
  - I_3=P_e X t_f : task completed.
  - I_4=P_f X t_h : interrupt to successor task.
  - I_5=P_c X t_d $\cup$ I_1=P_a X t_d $\cup$ I_1=P_r X t_d $\cup$ I_1=P_d X t_f are input arcs of the subnet formed dynamically.
- O is the set of output arcs, which are of the following types:
  - O_1=t_hXP_h: task not dependent on any other task.
  - O_2=t_cXP_c: agent assigned.
  - O_3=t_eXP_e: resource allocated.
  - O_4=t_fXP_r: resource released.
  - O_5=t_fXP_f: Task completed by agent.
  - O_6=t_fXP_a: agent released.
  - O_7=t_dXP_d: output arcs of the subnet formed dynamically.
- TOK is the set of color tokens present in the places of Petri net. So we can write, $TOK = \{TOK_1, TOK_2, \ldots, TOK_X\}$, where each TOK_i where $1 \leq\ i\ \leq\ X$, is associated with a function $assi_tok$ defined as:

  $assi_tok$: $TOK \rightarrow (CategoryXTypeXN)$, where, $Category$ = set of all categories of tokens in the system= $\{T, R, A\ \}$, $Type$ = set of all

types of each $category_i \in Category$ i.e. Type= $\{T \cup R \cup A\}$,
N is the set of natural numbers.

Let $assi_tok(TOK_i)$ = ($category_i$, $type_i$, n_i). The function $assi_tok$ satisfies the following constraints:

- $\forall TOK_i$ ($category_i$ = R)$\rightarrow$\{($type_{i\in}$ R) $\wedge$ ($1 \leq n_{i\leq}$ $inst_R(type_i)$)\}
- $\forall TOK_i$ ($category_i$ = A)$\rightarrow$\{($type_{i\in}$ R) $\wedge$ ($1 \leq n_{i\leq}$ $inst_A(type_i)$)\}
- $\forall$ TOK_i ($category_i$ = T) $\rightarrow$ \{($type_i$ $\exists$ T) $\wedge$ (n_i = 1) \}.

$assi_tok$ defines the category, type and number of instances of each token.

- F_n is a function associated with each place and token. It is defined as:

  F_n: $PXTOK \rightarrow (TIME \text{ X } TIME)$.

  For a token $TOK_k \in TOK$, $1 \leq k \leq x$,

  and place $P_i \in P$, $F_n(P_i, TOK_k) = \{(a_i, a_j)\}$, a_i is the entry time of TOK_k to place P_i and a_j is the exit time of TOK_k from place P_l. For a token entering and exiting a place multiple times, $|F_n(P_i, TOK_k)|$ = number of times TOK_k entered the place P_i.

6.2 Rules to construct Task Petri Nets

A Task Petri Nets can be created if the following information is provided:

- An initial set of tasks, identified by the interface agent, called $TASK$.
- The happened before relationships between the tasks, described in the form of Task Graph.
- The resource requirements of each task, $task_i \in TASK$.
- The capabilities of the agents present in the system.

Given this information, we can construct the Task Petri Nets by following rules:

- Construct place P_t. It will represent the user's query mapped to a single task.
- Construct places P_{hi}, P_{ci}, P_{ei}, P_{fi} and transitions t_{hi}, t_{ci}, t_{ei}, t_{fi} for each $task_i \in TASK$ identified by the interface agent.
- Construct places P_{ai} for each type of agent a_j, $1 \leq j \leq p$, $a_j \in$ A. The number of tokens in P_{aj} place is $inst_A(a_j)$.
- Construct places for each type of resource r_k, $1 \leq j \leq p$, $r_k \in$ R. The number of tokens in P_{rk} place is $inst_R(r_k)$. For each $task_i \in TASK$, add an arc from,
  - place P_t to transition t_{hi} and t_{hi} to place P_{hi}.
  - place P_{ci} to transition t_{ei} and t_{ei} to place P_{ei}.
  - place P_{ei} to transition t_{fi} and t_{fi} to place P_{fi}.
- For each $task_i$, if there are 'n' number of different type agents ($a_{j1}, \ldots, a_{jn}$) capable of performing it, construct 'n' number of t_{ci} transitions (t_{ci1}, $t_{ci2}, \ldots, t_{cin}$) and add arcs from P_{hi} to all of them and from the places ($P_{aj1}, \ldots, P_{ajn}$) to transitions ($t_{ci1}, \ldots, t_{cin}$) respectively. Add arcs from all t_{ci} transitions ($t_{ci1}, t_{ci2}, \ldots, t_{cin}$) to place P_{ci}.
- For all (r_k, n_k), where (r_k, n_k) $\in$ req_res($task_i$), add an arc(P_r X t_e) from P_{rk} to t_{ei}. The weight of the arc is equal to n_k. Add an output arc(t_f X P_r) from each t_{fi} to all P_{rk}. The weight of the arc is equal to n_k.
- From the task graph, if $task_i \rightarrow task_j$, then add an arc ($P_f$ X t_h) from place P_{fi} to transition t_{hj}.
- Create 'm' number of t_{di} transitions ($t_{di1}, t_{di2}, \ldots, t_{dim}$) and add an arc ($P_c$ X t_d) from P_{ci} to each t_{dij}, $1 \leq j \leq m$, if the agent assigned to $task_i$ wishes to divide $task_i$ into âmâ sub-tasks. Create a place for each of âmâ subtasks. These places will be categorized as P_d. Add arcs from each transition ($t_{di1}, t_{di2}, \ldots, t_{dim}$) to places ($P_{di1}, P_{di2}, \ldots, P_{dim}$) respectively.
- From each P_{dij} place, the subnet is created in a similar way as the subnet is created from P_{ci} place for $task_i$. The subnet consists of P_e, P_f places and t_e, t_f transitions all categorized as P_d and t_d respectively. The resource requirements and the happened before relationships between the newly created tasks are determined by the agent dividing the task. The petri nets is modified by adding such places and transitions dynamically. The arcs are added according to the rules described for a $task_i$.

6.3 Metrics for Performance Analysis

We define the following three metrics for performance analysis of the system. These are:

6.3.1 Total Execution Time

This metric gives the total execution time of the system.
$\forall\, TOK_i$, $assi_tok(TOK_i) = (T, type_i, n_i)$,
$F_n(P_h, (T, type_i, n_i)) = \{(a_p, a_q) \mid a_p$ is the time at which token entered place P_h and a_q is the time at which token left the place.$\}$.

The start time of execution is the time at which the first task started, i.e. the minimum of the entry times to P_h place of all tokens. So,
S_t(Starting Time) =
$\min\{a_p \mid (a_p, a_q) \in F_n(P_h,(T,type_i, n_i))\}$,
$\forall\, TOK_i$, $assi_tok(TOK_i) = (T, type_i, n_i)$.

The finish time of execution is the finishing time of the last task of the system. So,
F_t(Finish time)=
$\max\,\{a_p \mid (a_p, a_q) \in F_n(P_h, (T, type_i, n_i))\}$,
where $\forall\, TOK_i$, $assi_tok(TOK_i) = (T, type_i, n_i)$.

So, total execution time
$(T_{et}) = \max\,\{a_p \mid (a_p, a_q) \in F_n(P_h, (T, type_i, n_i))\}$-$\min\{a_p \mid (a_p, a_q) \in F_n(P_h, (T, type_i, n_i))\}$, $\forall\, TOK_i$, $assi_tok(TOK_i) = (T, type_i, n_i)\}$.

6.3.2 Resource Utilization

This metric determines to what extent each resource in the system has been utilized. For each resource token TOK_i, $assi_tok(TOK_i)$=$(R, type_i, n_i)$,
$F_n(P_r, (R, type_i, n_i))$=$\{(a_p, a_q) \mid a_p$
is the entry time of this token to place P_r, a_q is the exit time of this token from place $P_r\}$,
Total time for which a token resided at place P_r is given by:
T_{pr}= Σ $(a_p$ - $a_q)$, $\forall (a_p,a_q)$ $\forall$ $F_n(P_r, (R, type_i, n_i))$
Total amount of time the resource was utilized
= $(T_{et}$ - $T_{pr})/T_{et}$
= $[1$-$(T_{pr}/T_{et})]$

6.3.3 Agent Utilization

This metric determines to what extent each agent in the system has been utilized. For each agent token TOK_i, $assi_tok(TOK_i)$=$(A, type_i, n_i)$,
$F_n(P_a, (A, type_i, n_i))$=$\{(a_p, a_q) \mid a_p$ is the entry time of this token to place
P_a, a_q is the exit time of this token from place $P_a\}$,
Total time for which a token resided at place P_r is given by:
T_{ar}= Σ $(a_p$ - $a_q)$, $\forall (a_p,a_q)$ $\forall$ $F_n(P_r, (A, type_i, n_i))$
Total amount of time the agent was utilized
= $(T_{et}$ - $T_{ar})/T_{et}$
=$[1$-$(T_{ar}/T_{et})]$

7 Comparison of Task Petri Nets with other forms of Petri Nets

There are other forms of Petri nets avaialble but they have been found inefficient to model the task assignment process. A comparison of some of these with Task Petri Nets is given in Table 1. Task Petri Nets have been compared with Color Petri Nets, Agent Petri Nets and Predicate Transition Nets on the following areas

- Supports MAS
- Deals with Task Assignment
- Evaluates Performance
- Capacity to model interactions between agents

Table 1: Comparison of Task Petri Nets with other forms of Petri Nets

Type of Petri Nets	Supports MAS	Deals with Task Assignment	Evaluates performance	Capacity to model interactions between agents
Color Petri Nets/ Object Petri Nets	No	No	No	No
Agent Petri Nets	Yes	Yes	No	Yes
Predicate Transition Nets	Yes	Yes	No	No
Task Petri Nets	Yes	Yes	Yes	Yes

8 Case Study

8.1 Problem

Consider the scenario where the user submits a query Q to the system. The interface agent maps Q to a set of tasks. Let this set be $TASK$ = {a, b, c, d, e}. There are two types of resources $R = \{r_1, r_2\}$and three types of agents A= $\{a_1, a_2, a_3\}$. There are 2 instances of r_1 and r_2 each and 3 instances of a_1, a_2, and a_3 each. The capabilities of the agents and the resource requirements of the tasks are defined as:
$able(a_1)$={a };
$able(a_2)$={b, d}; $able(a_3)$={e}.
req_res(a)=$\{(r_2, 2)\}$;
req_res(b)=$\{(r_1, 2)\}$;
req_res(c)=$\{(r_1, 2), (r_2,2)\}$;
req_res(d)=$\{(r_2, 2)\}$;
req_res(e)= $\{(r_1, 1), (r_2, 1)\}$The happened before relationships between these tasks identified by interface agent are: a$\rightarrow$ b; a $\rightarrow$ c; b$\rightarrow$d; b$\rightarrow$ e; c$\rightarrow$ d

8.2 Model of the problem

8.2.1 Task Graph

Figure 1 given below represents the task graph for the problem described in previous sub-section. Given a set of happened before relationships, we construct the task graph as defined in the section 5.

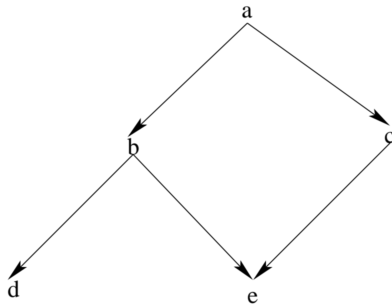

Figure 1: Task Graph for the problem

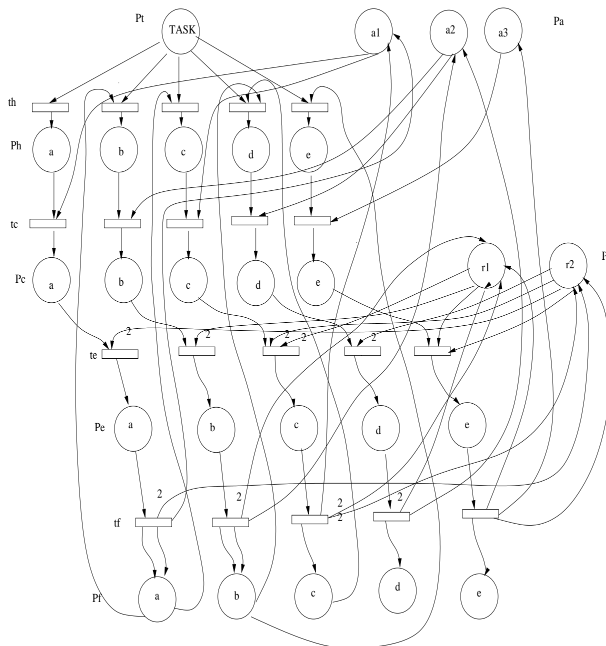

Figure 2: Task Petri Nets Model for the problem

8.2.2 Task Petri Nets

According to the rules described in section 6.2, we can draw the Task Petri Nets based on the given set of tasks identified by the interface agent, their requirement of resources and the capabilities of agent. Figure 2 shows the Task Petri Nets for the problem described above.

8.3 Simulation Results and Discussion

The above scenario is simulated in MATLAB. The value of one metric, total execution time has been calculated and its variation studied over three parameters 1) number of tasks, 2) number of resources and 3) number of agents. The duration of each task is assumed to be 10 sec. The following graphs show the change of execution time against the three parameters.

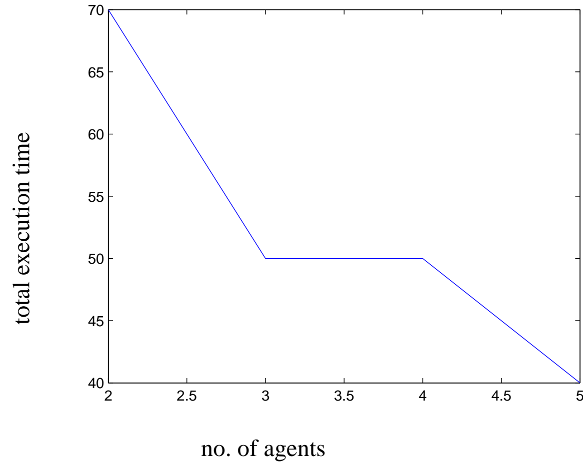

Figure 3: Graph showing total execution time against no. of agents

Figure 3 shows the change of total execution time against number of agents for a fixed number of tasks and resources. 7 tasks are taken and random happened before relationships are generated between them. There are 10 types of resources each having 4 instances. The resource requirements of the tasks are also generated randomly. There are 5 types of agents each having 2 instances initially. Their abilities are generated randomly. Keeping the number of tasks and the number of resources fixed, the number of instances of each agent type is increased by 1 along the X-axis. The total execution time for 2, 3, 4 and 5 number of instances of agents is plotted in Figure 3. The number of agents available in the system is a bottleneck as it determines the number of tasks executing concurrently. If a task is not dependent on any other task/tasks for execution, it should be assigned to an agent. But, if all the agents capable to perform that task are busy executing other tasks, the task cannot execute. So, the task has to wait until a capable agent is free. From the graph, it is observed that as the number of agents in the system increases, the execution time decreases or remains same. It is because as the number of agents in the system increases, the tasks which could not be assigned to agents due to their unavailability earlier are assigned due to an increase in their number. So, number of tasks executing concurrently increases which decreases the total execution time. The total execution time remains same when the system achieves the maximum possible concurrency or the concurrency depends on the availability of resources.

Figure 4 shows the change of execution time against

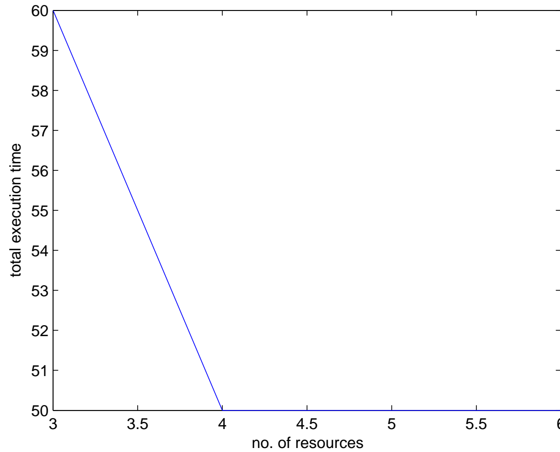

Figure 4: Graph showing total execution time against no. of resources

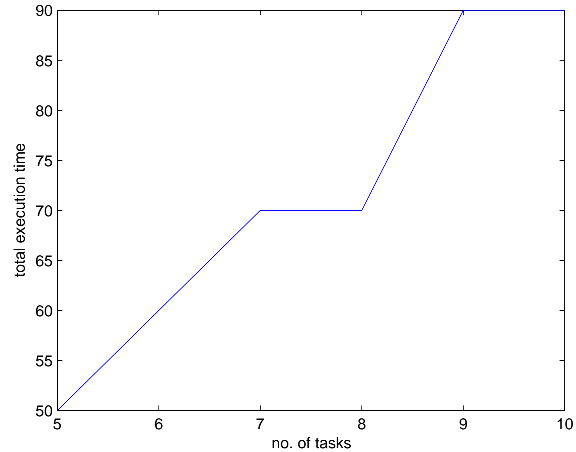

Figure 5: Graph showing total execution time against no. of tasks

number of resources for a fixed number of tasks and agents. 7 tasks are taken and random happened before relationships are generated between them. There are 5 types of agents each having 3 instances. Their abilities are generated randomly. There are 10 types of resources each having 3 instances initially. The resource requirements of the tasks are also generated randomly. Keeping the number of tasks and the number of agents fixed, we increase the number of instances of each resource type by 1 along the X-axis. The total execution time for 3, 4, 5 and 6 number of instances of resource types is plotted in Figure 4. The number of resources available in the system is also a bottleneck as it determines the number of tasks executing concurrently. If a task is not dependent on any other tasks for execution and it is assigned an agent, it should get the required resources to start execution. But, if all the resources required are not available, the task cannot execute. It has to wait until the requested resources are available. As we increase the number of resource instances in the system, the execution time decreases or remains same. The reason is that as we increase the number of resources in the system, the tasks which could not execute due to unavailability of resources can now execute due to their increased availability. So, number of tasks executing concurrently increases which decreases the total execution time. The execution time remains the same when the added resources are not utilized, i.e. the system already achieves maximum possible parallelism.

Figure 5 shows the change of execution time against number of tasks for a fixed number of resources and agents. 5 tasks are taken initially and random happened before relationships are generated between them. There are 5 types of agents each having 2 instances. Their abilities are generated randomly. There are 10 types of resources each having 2 instances. The resource requirements of the tasks are also generated randomly. Keeping the number of resources and the number of agents fixed, we increase the number of tasks by 1 along X-axis. When a new task is added to the system, it is either made independent or dependent on some other task/tasks randomly. The total execution time for 5, 6, 7, 8, 9 and 10 tasks is plotted in Figure 5. If the new task is dependent on previous task/tasks, then the execution time increases or remains same, otherwise it executes concurrently provided the agents and required resources are allocated. So, we get an increasing graph when we plot the total execution time against the number of tasks.

Figure 6 shows the change in total execution time against two parameters 1) number of tasks, 2) number of resources. There are 10 types of resources each having 5 instances initially. The resource requirements of the tasks are generated randomly. There are 5 types of agents each having 4 instances. Their abilities are generated randomly. The number of resources is increased by 1 along Y axis i.e. 5, 6, 7 and 8 and number of tasks is increased by 1 along X axis i.e. 5, 6, 7, 8, 9 and 10. The total execution time is plotted in Figure 6. It is clear from the graph that the execution time increases or remains same when the number of tasks increases while decreases or remains same when the number of resources increased.

Figure 7 shows the change in total execution time against two parameters 1) number of tasks, 2) number

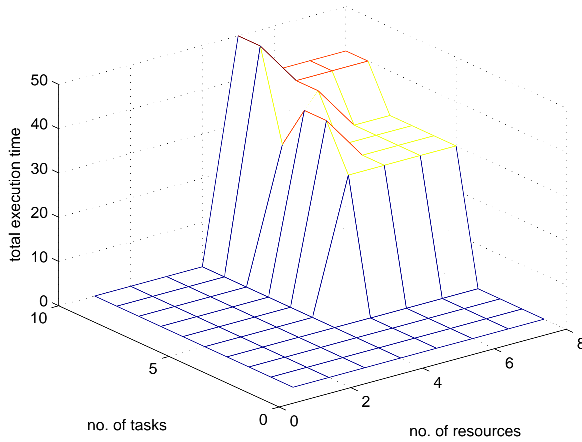

Figure 6: Graph showing total execution time against no. of tasks and no. of resources

of agents. There are 10 types of resources each having 3 instances. The resource requirements of the tasks are generated randomly. There are 5 types of agents each having 2 instances initially. Their abilities are generated randomly. The number of agents is increased by 1 along Y axis i.e. 2, 3, 4 and 5 and number of tasks is increased by 1 along X axis i.e. 5, 6, 7, 8, 9 and 10. The total execution time is plotted in Figure 7. Similar to the previous case, execution time increases or remains same when the number of tasks increases while decreases or remains same when the number of agents increases.

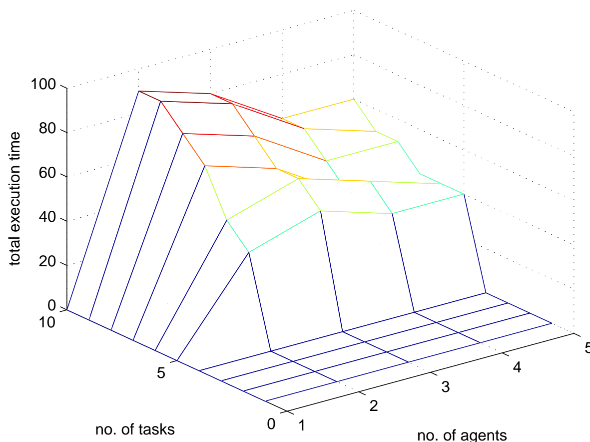

Figure 7: Graph showing total execution time against no. of tasks and no. of agents

Figure 8 shows the change in total execution time against two parameters 1) number of agents, 2) number of resources. There are 10 types of resources each having 2 instances initially. The resource requirements of the tasks are generated randomly. There are 5 types of agents each having 3 instances initially. Their abilities are generated randomly. The number of tasks is fixed at 5. The number of agents is increased by 1 along Y axis i.e. 3, 4, 5 and 6 and number of resources is increased by 1 along X axis i.e. 2, 3, 4 and 5. The total execution time is calculated and plotted in Figure 8. It is clear from the graph that the execution time decreases or remains same along both axes, when the number of agents and resources increases.

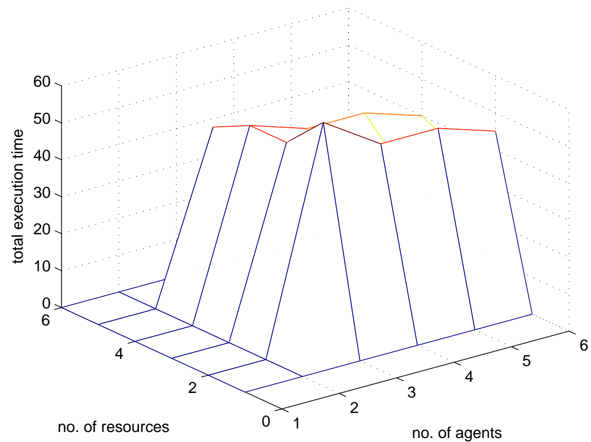

Figure 8: Graph showing total execution time against no. of agents and no. of resources

9 Conclusion

In this paper, a Task Petri Nets tool to address the task assignment problem in MAS is developed. It can represent agent autonomy and task delegation property. It can also evaluate the performance of the system using three metrics. This tool proves better than the existing tools on grounds as described in Table 1 to model such scenario. A scenario is simulated in MATLAB and the value of one of the three metrics is measured. From the results and discussion, we conclude that to reduce the total execution time, we have to increase the number of tasks executing concurrently. This can be done by increasing the number of agents or resources available in the system. Thus, the overall impact on the system can be analysed with the help of this tool. The future work is to develop a co-ordination mechanism

between agents to resolve resource conflicts to increase concurrency of tasks. It may so happen that some of the agents will crash while executing some tasks. So, the future work can also be to develop a fault tolerant system which will continue to work even if some of the agents crash while execution.

References

[1] Bai, Q., Zhang, M., and Zhang, H. A coloured petri net based strategy for multi-agent scheduling. In *Proceedings of the Rational, Robust, and Secure Negotiation Mechanisms in Multi-Agent Systems*, 2005.

[2] Celaya, J. R., Desrochers, A. A., and Graves, R. J. Modeling and analysis of multi-agent systems using petri nets. *Journal of Computers*, 4:981–996, 2009.

[3] Ferreira, P. R. and et. al et. al, F. S. Robocup rescue as multiagent task allocation among teams: experiments with task interdependencies. In *Autonomous Agents and Multiagent Systems*, 2009.

[4] Gelenbe, E. and Timotheou, S. Random neural networks with synchronized interactions. *Neural Computation, 20*, pages 2308–2324, 2008.

[5] Jensen, K. Colored petri nets – basic concepts, analysis methods and practical use. *Software engineering, Springer-Verlag, Berlin*, 1992.

[6] Jindian, S., Heqing, G., and Shanshan, Y. A coloured petri net model for composite behaviors in multi-agent system. In *Proceedings of the IEEE Conference on Cybernetics and Intelligent System*, 2008.

[7] Lenstra, J. K. and Tardos, D. B. S. E. Approximation algorithms for scheduling unrelated parallel machines. In *Proceedings of 28th Annual Symposium on Foundations of Computer Science*, 1987.

[8] Macarthur, K. S. and et. al et. al, M. V. Decentralised parallel machine scheduling for multi-agent task allocation. In *Proceedings of Fourth International Workshop on Optimisation in Multi-Agent Systems*, 2011.

[9] Macarthur, K. S. and et. al et. al, R. S. A distributed anytime algorithm for dynamic task allocation in multi-agent systems. In *Proceedings of Twenty-Fifth Conference on Artificial Intelligence (AAAI)*, 2011.

[10] Maheswaran, R. J., Pearce, J., and Tambe, M. A family of graphical-game-based algorithms for distributed constraint optimization problems. In *Coordination of Large-Scale Multi-agent Systems, Springer-Verlag, Heidelberg Germany*, pages 127–146, 2005.

[11] Marzougui, B., Hassine, K., and Barkaoui, K. A new formalism for modeling a multi agent systems: Agent petri nets. *J. Software Engineering and Applications*, 3:1118–1124, 2010.

[12] Modi, P. J. and et. al et. al, W. S. Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence Journal, 161*, pages 127–146, 2005.

[13] Murata, T. and Nelson, P. C. A predicate - transition net model for multiple agent planning. *Information Sciences*, pages 57–58, 361–384, 1991.

[14] Nilsson, N. J. Artificial intelligence: a new synthesis. In *Morgan Kaufmann Publishers Ltd.*, 1998.

[15] Padgham, L. and Winikoff, M. Developing intelligent agent systems- a practical guide. *John Wiley & Sons, Ltd.*, 2002.

[16] Petcu, A. and Faltings, B. Dpop: A scalable method for multiagent constraint optimization. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 266–271, 2005.

[17] Ramchurn, S. D. and et. al et. al, A. F. Decentralised coordination in robocuprescue. In *The Computer Journal 53(9)*, pages 1447–1461, 2010.

[18] Rouff, C. and et. al et. al, M. H. Agent technology from a formal perspective. *Springer-Verlag London Limited.*, 2006.

[19] Russell, S. J. and Norvig, P. Artificial intelligence: A modern approach. In *Pearson Education.*, 2003.

[20] Scerri, P. and et. al et. al, A. F. Allocating tasks in extreme teams. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multi-Agent Systems, ACM New York, NY, USA*, pages 727–734, 2005.

[21] Tamma, V., Aart, C., and et. al. et. al., T. M. An ontological framework for dynamic coordination. In *Proceedings of 4th International Semantic Web Conference. (ISWC 2005)*, 2005.

[22] Tanenbaum, A. Distributed operating systems. In *Pearson Education.*, 1995.

[23] Vidal, J. M. Fundamentals of multi agent systems, 2007.

[24] Weiss, G. Multi agent systems: a modern approach to distributed artificial intelligence. In *MIT Press*, 1999.

[25] Wooldridge, M. J. Introduction to multi agent systems, 2001.

[26] Xu, D., Volz, R. A., Ioerger, T. R., and Yen, J. Modeling and verifying multi-agent behaviors using predicate transition nets. In *Proc. of the 14th International Conference on Software Engineering and Knowledge Engineering*, pages 193–200. ACM Press, 2002.

[27] Zheng, X. and Koenig, S. Reaction functions for task allocation to cooperative agents. In *Proceedings of the International Joint Conference on Autonomous Agents and Multi-agent Systems AAMAS 08*, pages 559–566, 2008.