Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844
Vol. V (2010), No. 1, pp. 91-105

Design of Protocols for Task Administration in Collaborative Production Systems

H. S. Ko, S. Y. Nof

Hoo Sang Ko
School of Industrial Engineering,
Purdue University 315 N Grant St,
West Lafayette, IN 47907, USA
E-mail: ko@purdue.edu

Shimon Y. Nof
School of Industrial Engineering,
Purdue University 315 N Grant St,
West Lafayette, IN 47907, USA
E-mail: nof@purdue.edu

Abstract:
Customer-focused and concurrent engineering service systems process tasks more effectively as a result of the power of collaboration among multiple participants. In such environments, however, complex situations might arise that require decisions beyond simple coordination.Task Administration Protocols (TAPs) are designed as a control mechanism to manage complex situations in collaborative task environments. This article presents the design of TAPs for collaborative production systems in which tasks are performed by the collaboration of multiple agents. Three component protocols are found to constitute TAPs and are triggered at appropriate stages in task administration: 1) Task Requirement Analysis Protocol, 2) Shared Resource Allocation Protocol, and 3) Synchronization & Time-Out Protocol. A case study with TAPs metrics for task allocation in a collaborative production system is investigated to compare performance under TAPs, and under a non-TAP coordination protocol (which is considered to be simpler). In terms of task allocation ratio, the case study indicates that performance under TAPs is significantly better (up to 10.6%) than under the non-TAP coordination protocol, especially under medium or high load conditions. The advantage of TAPs can be explained by their design with relatively higher level of collaborative intelligence, addressing more complex control logic compared with non-TAP coordination protocols.
Keywords: rules, figures, citation of papers, citation of books, examples.

1 Introduction

In highly distributed networked systems, tasks arrive at agents in the system, and then are processed by the collaboration of agents that have the skills and capabilities to process the entire task or parts of it. The achievement of goals depends on how effectively each individual agent coordinates tasks with others to solve the given problems in a collaborative manner. There have been numerous research studies regarding collaborative problem solving in a decentralized way by multiple agents, e.g., optimization by collaborative swarm intelligence [22], and collaborative negotiation in global supply networks [29]. As systems and systems-of-systems become more complex, especially when attempting to improve performance via collaboration, the dynamic complexity of interactions among agents requires a higher level of collaborative intelligence. Thus, there should be an effective control mechanism to rationalize, coordinate, and harmonize tasks by exchanging information and decisions among collaborating participants.

Coordination protocols have been developed to achieve effective control by managing a given, known type of dependence among tasks, e.g., producer/consumer relationships, as defined in coordination theory [1, 2]. Application of coordination protocols in collaborative service systems includes multi-robot systems [3, 30], supply chain management [4, 29], and agent-based manufacturing systems [10, 28], to name a few. The operation of these systems is usually represented by collaborative work in multi-agent systems [6, 23], and various coordination protocols have been developed to provide the agents in the systems with the rules and the interaction procedures on how to cooperate and coordinate effectively. Many coordination protocols have been developed based on the framework of contract net protocol (CNP) [7], a market-based approach in which tasks are announced to agents and allocated to the agent that provides the best bid. Such decentralized approaches have been developed to achieve effective resource allocation in distributed systems and have shown good performance in terms of communication and computational efficiency, scalability, and flexibility [5, 8, 9], compared to centralized approaches [2, 7].

A critical limitation of coordination protocols, however, is that there exist events that cannot be handled by merely coordination protocols. These situations include:

1. The priority of tasks must be dynamically changed; and

2. There are complicated situations requiring decisions beyond simple coordination.

For example, consider tasks that are close to their deadline and need to be handled quickly with high priority over other, less urgent tasks. Resources need to be allocated for the urgent tasks first in order to meet the deadline. Later, if a more urgent task, e.g., resource failure requiring repair, or an emergency task, is generated, the previously scheduled tasks may be preempted by the new one so that it will not cause a significant damage to the system. Coordination protocols, however, do not handle such situations since they only allow fixed task allocation, i.e., the winner to whom the task was allocated is deemed to commit itself to the awarded task until it is completed. In order to overcome such limitations of coordination protocols, the protocols need to be able to indentify repeatedly the current state of the system and take proper actions to deal with complicated conditions. Such protocols, which assume the responsibility of making decisions actively and triggering timely actions so that the overall system's performance can be further improved, are defined as Task Administration Protocols (TAPs) [11, 20].

The purpose of this article is to design TAPs and analyze TAPs' advantage over non-TAP coordination protocols. This is more important when more distributed and decentralized networks of activities require collaboration, which increases the complexity of the problem. In this article, the design of TAPs for effective task allocation and administration is developed by addressing three basic elements in task administration: 1) Task, 2) Resource, and 3) Time. Three main reasons require the development of TAPs with abilities beyond conventional coordination protocols, as follows.

(1) Dynamic tasks re-prioritizing

When tasks enter the system, they need to be analyzed and sorted based on their priority, so that important or urgent tasks can be handled first. Priority of tasks is an essential consideration in task administration. With coordination protocols, priority is evaluated by managerial decision or prior assignment based on task type [1], or by pricing based on market-based approach [8, 9]. Tasks with higher priority receive more monetary value, so the priority is reflected in the level of funding. The limitation in the coordination protocols is that the priority of a task is predetermined and fixed by its price and deadline in the task description at the moment the task arrives. While this approach may be adequate in some cases, in real situations, however, the priority may change due to emergent events. Thus, dynamic change of priority along time should be considered.

(2) Resource availability

Appropriate resources need to be allocated by considering the task requirements, the status of resources and their schedules. A bidding procedure can be processed to obtain good resource allocation solutions using the CNP framework (e.g., [8, 10]).

(3) Time-out conditions

A resource serving a task needs to be monitored. If a task occupies (engages) a resource excessively for certain reasons, e.g., resource failure, or abnormal tasks, the task access to the resource should be timed-out so that the overall performance of all tasks' services does not seriously degrade by the given task over-occupying the resource. In order to prevent such wasteful situations, a time-out protocol with an appropriate threshold needs to be used. In several studies and applications, time-out protocols have been developed and proved to be effective [12, 14].

In order to deal with the three basic elements of administration, TAPs' design model requires three components as developed in this study. The rest of the article is organized as follows. Section 2 describes the definitions and structures of TAPs with the three component-protocols. Each of the components is described in detail in section 3. In order to illustrate the advantage of TAPs over simple coordination protocols, a case study of applying TAPs for a collaborative production system is presented in section 4. Finally, section 5 concludes the article.

2 Task Administration Protocols (TAPs) in A Coordination Network

An active middleware model of coordination network [15, 16] is used to define the model of TAPs. A coordination network (Co-net) is a network of autonomous agents that enables collaboration among the agents. Each agent exchanges information and execution to achieve its objectives while it tries to maximize the system (or system-of-systems) goals. The Co-net is defined as:

$$Co\text{-}net =< \pi, \tau, \alpha, \sigma > \tag{1}$$

where π: a set of agents in *Co-net*; π = {A1, A2,..., $A_{|\pi|}$};τ a set of tasks to be processed in *Co-net*; τ = {T1, T2,..., $T_{|\tau|}$};α: a set of activities performed by π to fulfill τ; α = {$Activity_1$, $Activity_2$,..., $Activity_{|\alpha|}$}; and σ: a set of control mechanisms used by π.

Coordination is defined as the process of managing dependencies between activities [1]. In the *Co-net*, dependence (δ) is a relationship between τ, π, and α, and defined as follows:

$$\{T_i \times \pi(T_i) \times \alpha(T_i) | T_i \in \tau\} \rightarrow \delta \in \triangle \tag{2}$$

where π(Ti): a set of agents who can process Ti; α(Ti): a set of activities or sub-tasks required to process Ti; $\triangle$: a set of dependence types. A particular dependence δ can be decided by analyzing tasks and evaluating certain factors in the system, e.g., task priority and time-out threshold.

In the *Co-net*, coordination protocols (CP) are designed to manage a particular dependence (δ) between tasks and resources. A coordination protocol is defined as follows:

Definition 1.

$$P_j = \{\delta, I, R, PA, S\} \in CP \tag{3}$$

where P_j: a particular coordination protocol for handling T_i (j=1,..., J; J is the number of CP); I: a set of initiators which initiate coordination activities; $I \subset \pi$; R: a set of responders which respond to initiators' requests; $R \subset \pi$ and $I \cap R = \emptyset$; PA: a set of parameters of the coordination protocol; S: a set of decision logic in each transition stage of the protocol; and CP is a set of coordination protocols.

TAPs are defined as a set of protocols which assume the responsibility of making decisions actively and triggering timely actions so that these decisions and triggers can improve the coordinated performance [11, 20]. When a task (T_i) or, in a broader sense, an event (e) occurs in the system, TAP mechanism should identify the dependencies between tasks and activate the appropriate protocol (P_j) between agents with proper parameter values (PA) for the protocol. In the Co-net, therefore, a TAP is defined as

follows:

Definition 2.

$$TAP = \{\sigma \cup P_j | \sigma(e|\delta \subset \triangle) \rightarrow P_j\} \tag{4}$$

where $\sigma$is a mechanism to select an appropriate protocol to handle the upcoming events; e is an event (e ∈ E; E: a set of events), e.g., arrival of a task, failure in processing a task, etc. In other words, the TAP is a control mechanism to handle an event by analyzing dependence between tasks and involved initiators and responders, as well as the parameters and decision logic required to deal with the event. According to the type of event and systems of interest, different task administration schemes need to be applied.

Theorem 1.(TAP-Set). CP is a subset of but not identical to TAP.

Proof. According to *Definition 1*, CP is a set of P_j's which handles tasks under certain dependence $\delta \subset \triangle$. By *Definition 2*, TAP includes P_j's as its elements but also contains the mechanisms (σ) to dynamically activate appropriate P_j to handle the upcoming events e based on the current dependence $\delta \subset \triangle$. Moreover, CP cannot handle all types of events in task administration shown in Table 1, unlike TAP.

Theorem 2. (TAP-Performance). Better TAP yields better performance than CP.

Proof. Let θ be a performance metric. Since CP can handle only predefined δ, even though the dependence changes from δ_t to δ_{t+1} along time t, $P^t\ (\delta_{t+1}) = P^{t+1}\ (\delta_{t+1})$, where P^t is a protocol activated at time t. Since TAP can trigger a more appropriate protocol and its parameters under the current dependence and event, one can find P^{t+1} such that $\theta(P^t\ (\delta_{t+1})) < \theta\ (P^{t+1}\ (\delta_{t+1}))$. Therefore, the overall performance $\sum_t \theta\ (P^t\ (\delta_t))$ under TAP (P^t variable) will be better than under CP (P^t fixed) along time. This will be illustrated in a case study in section 4.

There are three general elements in task administration: task, resource, and time. The general administration elements, their properties and protocol solutions, which has been found from existing literatures, are presented in Table 1. TAPs are composed of three component-protocols, each of which will be activated by the TAP mechanism under certain dependence and deals with the events corresponding to an administration element. When a task arrives at the system and is inserted to task queue, the task requirement analysis protocol (TRAP) is activated to analyze the task and assign its priority and dependence to other tasks and resources. Upon arrival at task queue, a task must be assigned to the best resource, which is decided by the shared resource allocation protocol (SRAP), based on the current status of resources. While processed, a task may occupy a resource excessively, preventing other tasks from being processed by the resource. If the time taken in the resource is beyond a certain threshold, the task needs to be returned to task queue so that other tasks can be processed at the resource. Sometimes, a task cannot be performed by its due date under the current schedule. In this case, the task may need to seek a resource which can complete the task by preempting the other task currently being processed at the resource. These procedures are controlled by the synchronization & time-out protocol (STOP). Each of the three protocols is explained in the subsequent sections.

The theorems presented above are illustrated with three examples in Table 2. In the examples, both TAPs and CPs are used to control multi-agent interactions in certain distributed production systems, but TAPs are more intelligent because of inclusion of one or more TAP components which are missing in CPs. In all the three examples, the performance under TAPs is better than under CPs as they are designed to dynamically handle complex tasks. Even though the TAPs include all three components, there could be a difference in performance between different TAP designs, as in the first example [21]. The three component protocols are explained in the next section.

Table 1: Administration elements, their related events and dependence, and TAP solutions and examples

Admin. Elements	**Task** (τ)	(Resource (R)	**Time (t)**)
Events (e)	Task arrival	Resource allocation	Excessive process time, urgent tasks
Dependence Analyses (δ)	1. Prioritize tasks 2. Identify task requirement/dependence (δ)	1. Optimal/effective allocation 2.Decentralized decision making	1. Synchronization 2. Uncertainty in processes & products
Protocols P_j	Task Requirement Analysis Protocol (TRAP)	Shared Resource Allocation Protocol (SRAP)	Synchronization & Time-Out Protocol (STOP)
Examples [Ref's]	Pricing of tasks [8-10, 17-18]; Priority assignment protocol [13,21]	Market-based resource allocation [8-10,16-18]	Time-out protocol [12-14]

Table 2: Illustration of TAP-Set and TAP-Performance

Example $[Ref's]$	TAP	TAP Component	CP	Performance Metrics (θ)	Results
[21]	TestLAN protocol 1) TAP1: adaptable 2) TAP2: non- 2) TAP2: non-adaptable	TRAP, SRAP, STOP	TestLAN protocol w/ FCFS	waiting time, flowtime	θ(TAP1) $>$ θ(TAP2) $>$ θ(CP)
[14]	Time-out protocol	STOP	Non-timeout protocol	flow time, service time	θ(TAP) $>$ θ(CP)
[16]	Viability-based Resource allocation protocol	SRAP	Resource allocation protocol w/o viability	profit, number of un-allocated tasks	θ(TAP) $>$ θ(CP)

3 Components of Task Administration Protocols

3.1 Task Requirement Analysis Protocol (TRAP)

A key objective in task administration is to allocate incoming tasks to appropriate resources continuously. When this allocation is needed, TRAP, which is a component-protocol of TAPs, is activated to handle the event. This protocol finds the dependence between tasks by their priority relationships. Whenever a new task T_i arrives, a task agent (TA) calculates the priority of T_i and previous tasks still in the system at time t by using a dynamic task priority evaluation function (pf(T_i, t)). If the priority of T_i is less than the priority of previous tasks still in the system, it is just added to the end of the task queue as in First In, First Out regime. Otherwise, the tasks in the queue need to be sorted by their current relative priority. From time to time, the task currently being processed may need to be preempted by an urgent task with relatively higher priority.

Each task has different requirements, such as the type of task, quantity (volume), deadline, estimated cost, etc. In general, a task can be defined as follows:

$$T_i = < type_i, gty_i, dd_i, v_i, PR_i(t) > \tag{5}$$

where $type_i$ is the class of T_i that requires a certain skill of a resource agent (RA); qty_i is the task amount involved in T_i that engages the capacity of a RA; dd_i is the latest time by which T_i must be served by an RA; v_i is the estimated value per unit of T_i; and PR_i(t) is the priority of T_i at time t. PR_i(t) represents the relative importance of T_i and the task that has higher priority should be served first. PR_i(t) is dynamically evaluated by using the priority evaluation function pf(T_i, t). Priority evaluation function needs to be of different form to correspond with specific applications. In its generic form of supply-demand networks, the priority evaluation function is defined as:

$$pf(T_i,t) = w_1 s_i + w_2 \frac{gty_i(v_i - c_i)}{\sum_j gty_j(v_j - c_j)} + w_3\{1 - \frac{dd_i - t}{\sum_j (dd_j - t)}\} + w_4(1 - \frac{gty_0^t}{gty_o}) + w_5 \sum\nolimits_k (1 - pe_k) \tag{6}$$

where w_n: weight of each factor ($0 \le w_n \le 1$; $\sum w_n$ = 1; n = 1, 2,..., 5); j: index of a task previously assigned to the agent; 0: index of the current task; s_i = 1 if $type_i$ = $type_0$, 0 otherwise; c_i: estimated unit cost to perform T_i by a server; qty_0^t: remaining quantity of task T_0 at t; k: index of tasks that will be decommitted due to T_i ($PR_i(t) > PR_k(t)$); pe_k: penalty of decommitment of T_k; $0 \le pe_k \le 1$. The implication of the five factors is as follows:

1) If T_i is the same type of task as the current task, its priority is high since setup is minimized or not needed. In some case, however, the opposite is required.

2) The relatively larger T_i's quantity and profit (value) are, the higher its priority.

3) A task relatively closer to its deadline has higher priority.

4) If the current task is not finished yet but almost is close to being done, it has a relatively higher priority.

5) After its priority re-assignment,$T_i$may cause some tasks, which have already been assigned to resources, to be decommitted if they have a lower priority. The penalty due to the decommitment needs to be considered for fair evaluation of T_i's priority.

To sum up, TRAP is triggered upon arrival of a new task at a TA, which receives tasks and identifies task type and analyzes task requirements, i.e., due date, sub-tasks and required resources. TA assigns a priority value to each of the tasks based on the priority evaluation function. Tasks will be sorted and re-sorted in task queue by their relative priority. After sorting its queue, TA announces the tasks to the RAs who are capable of processing the tasks, and a resource will be allocated to each task one by one

by SRAP, which is explained in the next section. The overall procedure in TRAP can be summarized as follows:

1)Task agent (TA) receives a task T_i.

2)Calculate priority of tasks by Eq. (6).

3)Sort the tasks in task queue by PR_i(t).

4)Activate SRAP.

3.2 Shared Resource Allocation Protocol (SRAP)

After tasks are analyzed and their priorities are assigned, SRAP is activated to find the best resources for the tasks. Each of the resources in the system is managed by a resource agent (RA). TA announces the first task (with the highest priority) in task queue to RAs which are capable of processing the task. Each RA calculates the bid based on expected waiting time in the queue for the resource. TA collects the bids and selects the RA with the best bid (the lowest cost). The steps in SRAP are as follows:

1) TA announces T_i to RA_r where r is the index of RA which is capable of processing T_i.

2) RA_r sorts its queue including T_i by the priority of the tasks in the queue.

3) RA_r calculates the bid for T_i by using the following equation:

$$b_{r^i} = c_r \sum_{k=1}^{m_r} \mu_{pt_k} + cs_r \tag{7}$$

where b_{r^i} is the bid by RA_r for T_i; c_r is the cost of RA_r per unit time; k is the index of tasks in the queue of RA_r (k = 1,..., m_r);μ_{pt_k} is mean processing time of T_k; and cs_r is setup cost of RA_r, which is added only if the next task is of different class from the previous task.

4) RA_r submits b_{r^i} for T_i to TA.

5) TA selects RA_{r^*} where r^* = arg min_r b_{r^i}.

6) TA assigns T_i to RA_{r^*}. T_i enters the queue of RA_{r^*} with expected cost $b_{r^{*i}}$.

The policy applied in the above protocol pursues earliest completion time, which is reasonable when the effective cost of tasks is higher compared to the effective (variable) cost of resources. If the effective cost of resources is higher, relatively high utilization of resources may be preferred. In this case, a different policy, i.e., minimizing the total idle time of required resources, needs to be applied.

3.3 Synchronization and Time-Out Protocol (STOP)

From time to time, the current task being served by a resource needs to be timed-out and return to task queue of TA. This situation is triggered and handled by STOP, which is activated in the following cases.

A. Excessive resource occupation

Since a task with faults needs to be reworked by the resource, it may occupy the resource excessively even while idling it, and cause other tasks to be delayed. STOP checks if the current task uses the resource more than certain time-out threshold, which is calculated as follows [13]:

$$to_i = \mu_{pt_i} + 2\sigma_{pt_i} \tag{8}$$

where to_i is the time-out threshold for T_i, and σ_{pt_i} is standard deviation of processing time for T_i. (Two standard deviations are assumed for simplicity here, but another coefficient can be used.) Even if the task has been occupying the resource beyond the time-out threshold, the task remains at the resource's service in the following cases:

1) No other task is waiting in the queue.

2) The current task T_i may be late if timed-out. If dd_i - t $< ept_i$, where t is the current time and ept_i is the extra processing time needed for T_i, T_i remains in the resource.

B. Preemption by an urgent task

Even though the current task Ti has been served by the resource for less than the time-out threshold, a task Tk in the queue may be late if it is not served until the current process is completed. This problem may be solved by preempting Ti and serving the urgent task Tk first. The protocol logic can be summarized as follows:

1) Check $st_k = dd_k$ - t - μ_{pt_i} of the T_k waiting in the queue of the resource, where st_k is the slack time for T_k, i.e., the remaining time until dd_k.

2) If $st_k < 0$ and $dd_i > ept_i$, stop T_i and process T_k.

3) Once T_k is completed, resume T_i.

To sum up, TAPs are composed of three inter-related protocols: TRAP, SRAP, and STOP. Upon arrival of tasks, TRAP is activated to analyze the task requirements and assign their priority. Next, SRAP is activated to select the best resources based on their current workload. During processing tasks, STOP is activated to monitor if the current task in service needs to be timed-out because of its excessive use of the service or preempted by another urgent tasks. The overall protocol logic is shown in Figure 1. The following case study illustrates their application.

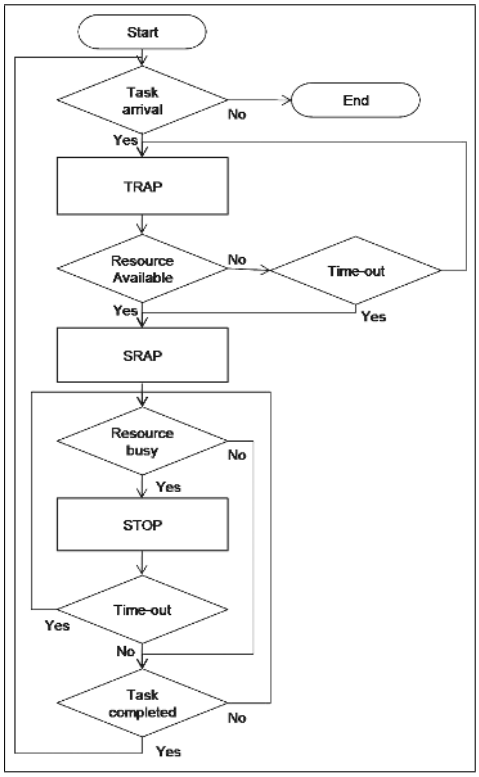

Figure 1: Overall logic of task administration protocols with three component-protocols

4 A Case Study: Collaborative Production Systems

In order to analyze the advantages of TAPs over non-TAP coordination protocols, this section illustrates a case study of TAPs application for task administration in a collaborative production system. A collaborative production system is defined as a distributed network of production resources in which re-

sources can communicate and coordinate with each other to process tasks in the system in an effective and efficient manner, based on collaborative resource sharing. A good example of collaborative production systems is TestLAN (Testing Local Area Network), which is a local area network integrating distributed test operations in manufacturing facilities [13]. TestLAN was developed to increase throughput and reduce the waiting time for testing by integration and communication of distributed testing servers and clients. In TestLAN, products are tested by shared resources, and those with faults are reworked and retested until all the faults are eliminated. Efficient testing resource allocation is difficult since the testing process is highly variable due to design changes, new quality control requirements, and occasionally faulty manufacturing processes.

In collaborative production systems like TestLAN, priority of tasks can be evaluated by their slack time. In TestLAN, assume only the third term in Eq. (6), regarding due date of the task, is effective in priority evaluation, and for simplicity assume every task is of the same importance and the objective is to complete as many tasks as possible in time. Hence, priority of a task in TestLAN is evaluated by a time-based cost of each task as follows:

$$st_i = dd_i - E[tt_i] \tag{9}$$

where tt_i is testing time of T_i. $E[tt_i]$ can be calculated by using the following equation [13]:

$$E[tt_i] = E[pt_i + \sum_{r=1}^{rn_i} rt_{ir}] = \mu_{pt_i} + \mu_{rn_i} \cdot \mu_{rt_i} \tag{10}$$

where pt_i is processing time of T_i; rn_i is number of reworks for T_i; rt_{ir} is rework time for r-th faults occurring in T_i; and μ_{pt_i} , μ_{rn_i} , and μ_{rt_i} are mean of processing time, number of rework, and rework time for the tasks of the same class as T_i, respectively. In order to reflect the rework in TestLAN, Eq. (7) and (8) need to be modified as follows:

$$b_r^i = c_r \cdot \sum_{k=1}^{m_r} (\mu_{pt_k} + \mu_{rn_k} \cdot \mu_{rt_k}) + cs_r \tag{11}$$

$$to_i = \mu_{pt_i} + 2\sigma_{pt_i} + \mu_{rn_i}(\mu_{rt_i} + 2\sigma_{rt_i}) \tag{12}$$

TAPs for the TestLAN function as follows. TRAP assigns a higher priority to a task with a lower slack time. Tasks are sorted in task queue by their priority and a resource who submits the best bid will be allocated to each task one by one by SRAP. In SRAP, each testing resource agent can sort the tasks in its own queue plus the new task to find the best bid. During processing the current task by each resource, time-out conditions are checked by STOP and the previously assigned tasks may be preempted if any of the two time-out preemption conditions are met. For this case study, two TAPs are developed: TAP1 and TAP2. Both TAP1 and TAP2 are composed of the three component-protocols. TAP1 considers, however, only time-out condition B in section 3.3 above, while TAP2 considers both time-out conditions A and B. A coordination protocol, CP, which is considered to be a subset of TAP, is used to compare performance under the various protocols. The logic of CP is similar to the one of SRAP, except the procedure of re-sorting tasks in resources' queue in SRAP is not included, as typical in traditional coordination protocols.

A simulation analysis is designed with two classes of tasks with different levels of urgency. Tasks are generated randomly and processed by three servers. The simulation parameters are shown in Table 3. Performance under the three protocols is measured by 1) task allocation ratio, TAR, and 2) weighted TAR, WAR, defined as follows:

$$TAR = 1 - \frac{\overline{T}}{|\tau|} \tag{13}$$

Table 3: Simulation parameters

Parameter	Value
Number of servers	3
Task class	Normal: dd_i = 600 sec, pt_i = 60 sec (80%) Urgent: dd_i = 180 sec, pt_i = 60 sec (20%)
Interarrival time	low: exp(μ = 25 sec); medium: exp(μ=20sec); high: exp(μ = 15 sec) where ? is the mean of interarrival time
Weight of pf(T_i, t)	w_3 = 1
Simulation length	T_E = 8 hrs
Rework rate	5%
Warm-up time	T_o = 10 min
Replication	rep = 10
Treatment	TAP1 = TRAP, SRAP, STOP1; TAP2 = TRAP, SRAP, STOP2; CP SRAP
Performance measures	TAR and WAR

Table 4: Java classes in TIE/TAP simulator

Sim	Supports the entire simulation, e.g., simulation initialization, termination, simulation clock, data collection
Model	Simulation modeling and execution, e.g., TestLAN model and parameters
Source	Random task generation
TA	Task agent which collects and announce tasks
RA	Resource agent which bids for and processes tasks
Task	Task definition and requirements
TRAP	Task Requirement Analysis Protocol
SRAP	Shared Resource Allocation Protocol
STOP	Synchronization & Time-Out Protocol

$$WAR = \frac{\sum_{i=1}^{|\tau|} \sum_{r=1}^{R} PR_i \cdot T_i^r}{\sum_{i=1}^{|\tau|} PR_i} \tag{14}$$

where $\overline{T} = |\tau| - \sum_{i=1}^{|\tau|} \sum_{r=1}^{R} |T_i^r|$;, r is the index of RA, $r = 1, ..., R$; T_i^r= 0 when T_i is neither assigned to RA_r nor completed within dd_i; 1 otherwise. While TAR does not consider the priority of tasks, WAR is used to measure the performance with consideration of task priority, i.e., assign more weights to relatively higher priority tasks when evaluating TAR.

The simulation is developed with a protocol evaluation tool, called TIE/TAP, which is implemented with Java, so as to compare the performance under three protocols: 1) TAP1, 2) TAP2, and 3) CP. The Java classes in TIE/TAP and their descriptions are listed in Table 4.

TAR results (Figure 2 and Table 5) are shown under low operational load (μ = 25; system capacity is sufficient to process all the tasks), medium (μ = 20; the number of tasks is close to the system capacity), and high (μ = 15; the number of tasks is beyond the system capacity). WAR results (Figure 3 and Table 5) are shown for the same load conditions. The observed results were tested for statistical difference by t-test as shown in Tables 6 and 7. In every case, it is found that TAP2 performs significantly better

Table 5: Simulation results

Protocol	Task Allocation Ratio low/medium/high[%]	Weighted Task Allocation Ratio low/medium/high[%]
TAP1	100/95.7/72.8	100/90.5/52.1
TAP2	100/97.4/74.6	100/94.2/54.0
CP	98.8/92.8/73.0	98.1/83.6/49.9

than TAP1 and than CP. Except for one case (TAR under high load condition), TAP1 is also significantly better than CP.

Under the low load condition, all three protocols studied perform similarly (TAR = 100% for TAP1 and TAP2, and 98.8% for CP). Under the medium load, the difference between the protocols increases. TAP2 yields the best TAR (97.4%), TAP1 – 95.7%, CP – 92.8%. Under the high load, TAP2 is also best (TAR = 74.6%), CP – 73.0%, TAP1 – 72.8%. Under every condition, TAP2 performs better than TAP1 and CP. TAP1 performs better than CP under the low and the medium load, but slightly worse under the high load. This situation results from the fact that TAP1 (and TAP2) considers priority of tasks by TRAP, tries to allocate the tasks with higher priority first, and even preempts the previously assigned tasks by STOP for the sake of the urgent tasks. Thus the TAR, which does not consider the priority of tasks, can be reduced under TAP1 (and TAP2) due to the failed tasks with low priority.

When WAR is considered (Table 5 and Figure 3), however, the performance of TAP1 is significantly better than CP under the high load. TAP2 is again the best (WAR = 54.0%), and TAP1 (WAR = 52.1%) also performs significantly better than CP (WAR = 49.9%). Under the medium load, WAR is 94.2%, 90.5%, and 83.6% for TAP2, TAP1, and CP, respectively. Therefore, when WAR is used to measure performance with consideration of task priority, the difference between TAPs and CP becomes larger under all conditions. This difference is expected from the fact that when tasks' priority changes dynamically, CP are not designed to track and respond to those changes.

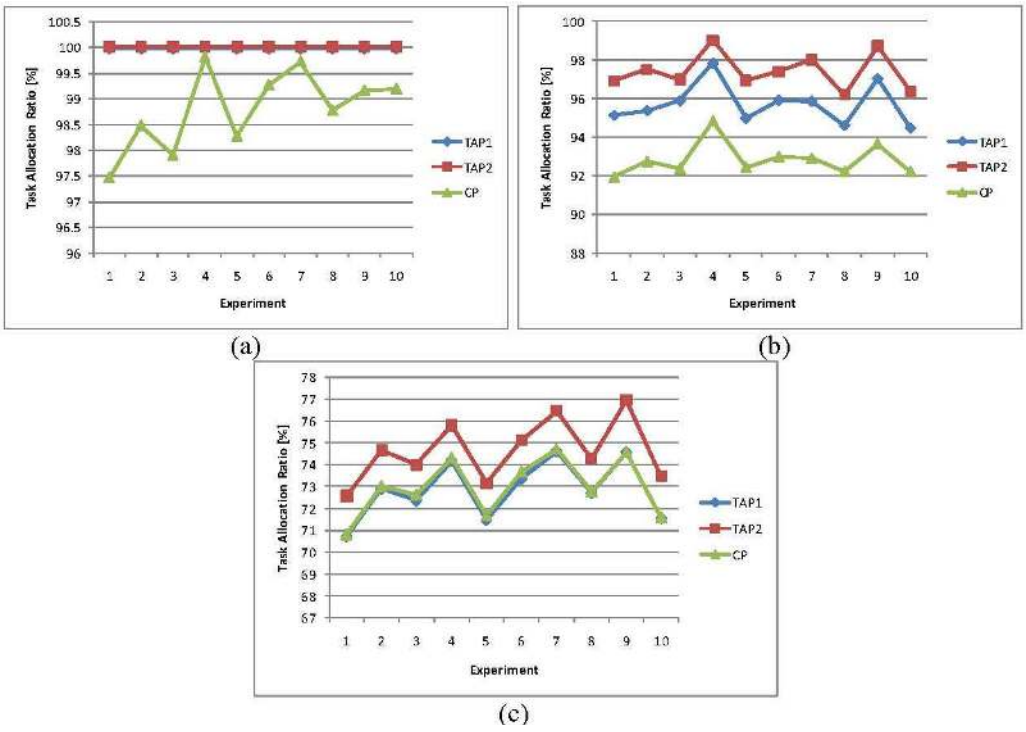

Figure 2: TAR under (a) low (μ = 25), (b) medium (μ = 20), and (c) high (μ = 15) load condition

Table 6: t-test for *WAR* of TAP2 and CP

Treatment	Mean		Std. Dev.		t		$t_{1-\alpha,v}$
	$\mu = 20$	$\mu = 15$	$\mu = 20$	$\mu = 15$	$\mu = 20$	$\mu = 15$	$\alpha = 0.05, v = 18$
TAP2	0.942	0.540	0.0198	0.0127	11.444	7.3414	1.734
CP	0.836	0.499	0.0192	0.0112			

Table 7: t-test for WAR of TAP1 and TAP2

Treatment	Mean		Std. Dev.		t		$t_{1-\alpha,v}$
	$\mu = 20$	$\mu = 15$	$\mu = 20$	$\mu = 15$	$\mu = 20$	$\mu = 15$	$\alpha = 0.05, v = 18$
TAP1	0.905	0.521	0.0219	0.0101	3.774	3.641	1.734
TAP2	0.942	0.540	0.0198	0.0127			

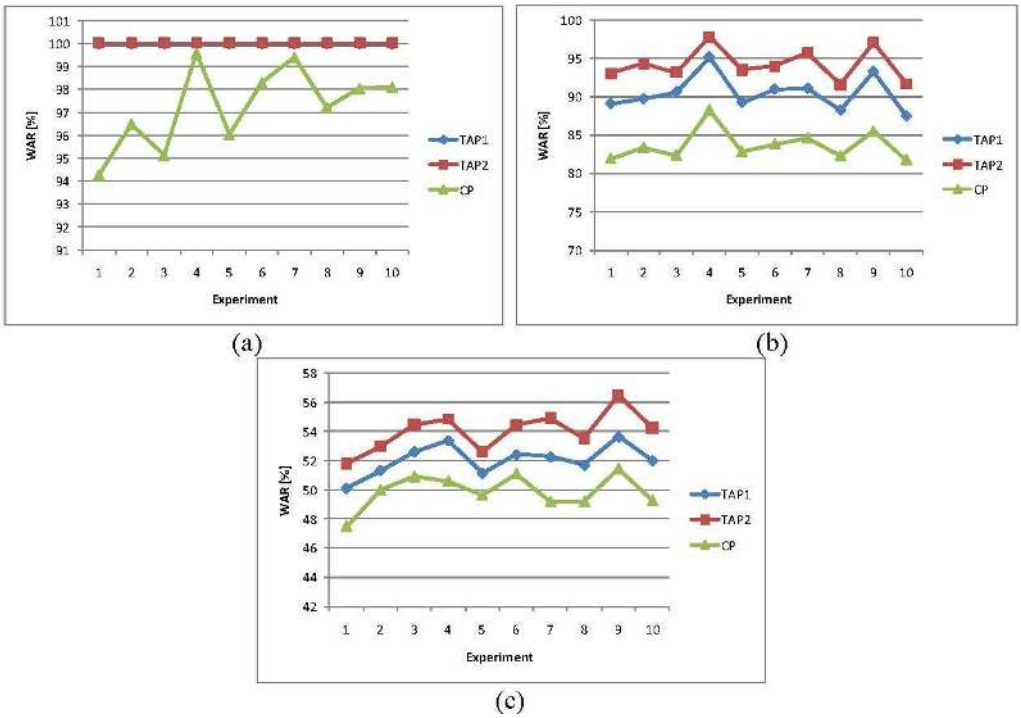

Figure 3: WAR under (a) low (μ = 25), (b) medium (μ = 20), and (c) high (μ = 15) load condition

In the studied case, the illustrated observations imply: (1) systems operating under TAPs perform better than under non-TAP CP, since TAPs can effectively handle the complicated situations that CP cannot (θ(CP) $<$ θ(TAP); see Table 6); and (2) even among TAPs, there can be better designed TAPs depending on the applications and logic applied (θ(TAP1) $<$ θ(TAP2); see Table 7).

5 Conclusions

The design of TAPs for task administration in a collaborative production system is investigated in this article. TAPs are designed as a control mechanism that can manage complicated situations in the collaborative task workflow environment. In order to overcome certain limitation of coordination protocols, TAPs are designed with three component-protocols, i.e., TRAP, SRAP, and STOP, each of which deals with inter-related aspects of task administration, including task, resource, and time. A case study of

applying TAPs for TestLAN, as an example of collaborative production system, is developed to illustrate design of TAPs and show the advantage of TAPs over non-TAP CP. The simulation, implemented based on the TIE/TAP Java-based simulator, show that TAPs perform significantly better than other non-TAPs, in particular under medium or high load conditions (up to 10.6% in terms of WAR). This advantage results from the fact that the three protocol combinations in TAPs dynamically interact to consider the dynamic priority of tasks and the current situations and conditions in tasks and resources. Thus, TAPs can address a higher level of collaborative intelligence compared to non-TAP CP. Finally, the results imply that (1) TAPs are better than non-TAP CP under certain conditions; and 2) there can be better design of TAPs even among TAPs to increase system performance.

The logic in the protocols is designed to fit the given case study. Although the general structure of protocols can be followed in order to handle the complicated situations in collaborative tasks/resources networks, for better effectiveness the protocol logic needs to be context-specific. Different applications require different decision policies, heuristics and logic, and they should be reflected in the protocol logic, as illustrated in the case study. In order to obtain better performance, the logic and the parameters in the protocol should be carefully selected and modified. For example, the system performance in the case study can be significantly affected by parameters such as time-out threshold. This recommendation is one of the challenging issues in protocol design, and some research has tried to address this issue by service-oriented protocol adaptation [21]. Further research is currently ongoing to develop and improve TAP logic with protocol adaptation for other applications, e.g., TAPs for collaborative intelligence in research activities [19], in which TAPs should be developed to manage collaborative workflow between distributed research groups and enhance their collaboration. Other similar, potential areas of TAPs application include collaborative intelligence in e-Learning environment [24], knowledge management in enterprise portal [25], and e-Service collaboration protocols [26].

Bibliography

[1] T.W. Malone, K. Crowston, The Interdisciplinary Study of Coordination, ACM Computing Surveys, 26(1):87-119, 1994.

[2] K. Crowston, J. Rubleske, J. Howison, Coordination Theory: A Ten-Year-Retrospective, Human-Computer Interaction and Management Information Systems: Foundations, P. Zhang, D. Galletta (eds.), M. E. Sharpe, Inc., 2006.

[3] B. Gerkey, M.J. Mataric, Sold!: Auction Methods for Multi-robot Control, IEEE Transactions on Robotics and Automation, 18(5):758-768, 2002.

[4] R.Y.K. Fung, T. Chen, A Multiagent Supply Chain Planning and Coordination Architecture, International Journal of Advanced Manufacturing Technology, 25(7):811-819, 2005.

[5] W. Shen, D.H. Norrie, Dynamic manufacturing scheduling using both functional and resource related agents, Integrated Computer-Aided Engineering, 8(1):17-30, 2001.

[6] M. Wooldridge, N.R. Jennings, Intelligent Agents: Theory and Practice, The Knowledge Engineering Review, 10(2):115-152, 1995.

[7] R.G. Smith, The Contract Net Protocol: High-level Communication and Control in A Distributed Problem Solver, IEEE Transactions on Computers, 29(12):1104-1113, 1980.

[8] S.S. Fatima, M. Wooldridge, Adaptive Task and Resource Allocation in Multi-agent Systems, Proceedings of the Fifth International Conference on Autonomous Agents, Montreal, Canada, 537-544, 2001.

[9] M.P. Wellman, E. Walsh, P.R Wurman, J.K. MacKie-Mason, Auction Protocols for Decentralized Scheduling, Games and Economic Behavior, 35(1-2):271-303, 2001.

[10] K. Ertogral and S. D. Wu, Auction-theoretic coordination of production planning in the supply chain, IIE Transactions, 32(10):931-940, 2000.

[11] C.Y. Huang, S.Y. Nof, Evaluation of Agent-based Manufacturing Systems Based on A Parallel Simulator, Computers and Industrial Engineering, 43(3):529-552, 2002.

[12] K. Esfarjani, S.Y. Nof, Client-server Model of Integrated Production Facilities, International Journal of Production Research, 36(12):3295-3321, 1998.

[13] N.P. Williams, Y. Liu, S.Y. Nof, TestLAN Approach and Protocols for The Integration of Distributed Assembly and Test Networks, International Journal of Production Research, 40(17):4505-4522, 2002.

[14] J. Peralta, J., P. Anussornnitisarn, S.Y. Nof, Analysis of A Time-out Protocol and Its Applications in A Single Server Environment, International Journal of Computer Integrated Manufacturing, 16(1):1-13, 2003.

[15] P. Anussornnitisarn, Design of Active middleware protocols for coordination of distributed resources, PhD dissertation, Purdue University, 2003.

[16] P. Anussornnitisarn, S.Y. Nof, O. Etzion, Decentralized Control of Cooperative and Autonomous Agents for Solving The Distributed Resource Allocation Problem, International Journal of Production Economics, 98(2):114-128, 2005.

[17] A.J. Arauzo, J.M. Galan, P. Javier, A. Lopez-Paredes, Multi-agent Technology for Scheduling and Control Projects in Multi-project Environments. An Auction Based Approach, Inteligencia Artificial, 42:12-20, 2009.

[18] Y.H. Lee, S.R.T. Kumara, K. Chatterjee, Multiagent Based Dynamic Resource Scheduling for Distributed Multiple Projects Using A Market Mechanism, Journal of Intelligent Manufacturing, 14:471-484, 2003.

[19] H.S. Ko, S.Y. Nof, Design of Collaborative e-Service Systems, Introduction to Service Engineering, G. Salvendy, W. Karwowski (eds.), John Wiley & Sons, Inc., 2009.

[20] J.D. Velasquez, S.Y. Nof, Collaborative e-Work, e-Business and e-Service, Springer Handbook of Automation, S.Y. Nof (ed.), Springer, 2009.

[21] N.P. Williams, Y. Liu, S.Y. Nof, Analysis of Workflow Protocol Adaptability in TestLAN Production Systems, IIE Transactions, 35(10):965-972, 2003.

[22] R.I. Lung, D. Dumitrescu, Collaborative Optimization in Dynamic Environments, International Journal of Computers, Communications, and Control, 1(Suppl), 2006.

[23] C. Kolski, P. Forbrig, B. David, P. Girard, C.D. Tran, H. Ezzedine, Agent-Based Architecture for Interactive System Design: Current Approaches, Perspectives and Evaluation, Lecture Notes in Computer Science, 5610:624-633, 2009.

[24] I. Moisil, A Model of the Student Behaviour in a Virtual Educational Environment, International Journal of Computers, Communications and Control, 3(Suppl):108-115, 2008.

[25] M. Guran, Knowledge Management using Intranets and Enterprise Portals, International Journal of Computers, Communications and Control, 3(Suppl):75-81, 2008.

[26] G. Kramler, E. Kapsammer, W. Retschitzegger, G. Kappel, Towards Using UML 2 for Modelling Web Service Collaboration Protocols, Interoperability of Enterprise Software and Applications, D. Konstantas et al (eds.), Springer, 2006.

[27] F.G. Filip, G. Neagu, D.A. Donciulescu, Job Shop Scheduling Optimization in Real-time Production Control, Computers in Industry, 4(4):395-403, 1983.

[28] J.M. Frayret, S. D'Amours, B. Montreuil, Coordination and Control in Distributed and Agent-based Manufacturing Systems, Production Planning and Control, 15(1)42-54, 2004.

[29] J. Jiao, X. You, A. Kumar, An Agent-based Framework for Collaborative Negotiation in The Global Manufacturing Supply Chain Network, Robotics and Computer-Integrated Manufacturing, 22(3):239-55, 2006.

[30] L. Iocchi, D. Nardi, M. Piaggio, A. Sgorbissa, Distributed Coordination in Heterogeneous Multi-robot Systems, Autonomous Robots, 15(2):155-168, 2003.

Hoo Sang Ko is a Senior Researcher of PRISM Center and a doctoral candidate in the School of Industrial Engineering at Purdue University. He received his B.S. in 1999 and M.S. in 2003 in Mechanical Engineering at Seoul National University in South Korea. After graduation, he worked as an R&D engineer at Digital Appliance Business of Samsung Electronics. His research interests include design of collaboration support systems, task administration protocols for control of collaborative e-Work, and facility sensor networks.

Shimon Y. Nof is Professor of Industrial Engineering and director of the NSF-industry supported PRISM Center (Production, Robotics and Integration Software for Manufacturing & Management) at Purdue University. The PRISM Center was established in 1991, and its motto is "Knowledge through information, wisdom through collaboration." In 2001, PGRN, PRISM Global Research Network was established with affiliate labs and centers throughout the world. Professor Nof was awarded an Honorary Doctorate in Engineering from the University of Sibiu, Romania in 2007, when Sibiu served as the European Capital of Culture.