A Key-based Coordination Algorithm for Dynamic Readiness and Repair Service Coordination[*]

Tom Wagner Valerie Guralnik John Phelps
Honeywell Laboratories
3660 Technology Drive, MN65-2600
Minneapolis, MN 55418
Tom.Wagner@honeywell.com

ABSTRACT

This paper describes an agent application for the coordination of aircraft repair, refit, refuel, and rearm teams in a dynamic setting. The paper also presents a new algorithm for dynamic distributed service team coordination and compares its performance to an optimal centralized service team scheduler.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed AI—*Coherence and Coordination, Multi-Agent Systems*

General Terms

Algorithms, Experimentation

Keywords

Coordination, Multi-Agent Systems, Intelligent Agents, TAEMS, Teams, Real-Time, Dynamic

1. INTRODUCTION

Aircraft returning from an engagement need refueling and potentially need new ordinance and repairs. The implications are that multiple different service crews, that have different capabilities and require different resources, must coordinate to effectively prepare the aircraft for another mission. For instance, it may not be desirable to service the engines while new ordinance is being loaded. Similarly it may not be possible to service the engines while refueling takes place. In contrast, it may be possible to overlap some activities, e.g., replacing the cockpit avionics while refueling the aircraft.

[*] Effort sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Office of Naval Research under agreement number N00014-02-C-0262 and by Honeywell International under project number I10105BB4. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), Office of Naval Research, the U.S. Government or Honeywell International.

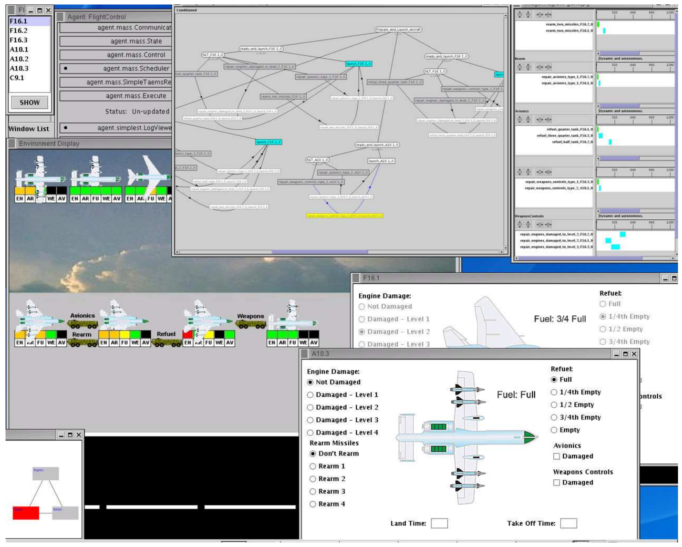


Figure 1: The Simulation Environment and Status Display

In this paper, we explore the use of agent technologies to coordinate aircraft service teams and present a new TÆMS (Task Analysis Environment Modeling and Simulation) [4] coordination algorithm used to coordinate service team activity. The performance of the algorithm is compared to a centralized scheduling oracle that generates optimal schedules for the teams – though the centralized scheduling problem is exponential, the oracle provides a good basis for comparison on smaller problem instances. A screen image of the application is shown in Figure 1. Note the "busy" task structure in the center of the screen – it is part of the centralized coordination problem that the agents solve through distributed and local reasoning with partial views (the centralized view is that of the simulation environment). In the following sections, we specify the problem space, define a new key-based coordination algorithm, and compare the algorithm to the centralized oracle.

2. DYNAMIC AIRCRAFT READINESS

In this project we simulate aircraft returning from an engagement and needing repairs / readiness operations to be performed. Three types of aircraft are modeled: F16s, A10s, and C9 surveillance craft. When an aircraft returns it is potentially in need of (to varying degrees): 1) fuel, 2) missiles, 3) repairs to engines, 4) repairs to cockpit avionics, or 5) repairs to cockpit weapons controls.[1] Each incoming

[1]Note that this is a small subset of the possible items needing service or inspection – this simplification is along a dimension that does not greatly impact the utility of the coordination algorithm.

	Repair Engines	Repair Avionics	Repair WeapCtrl	Refuel	Rearm
Engines				NLE	NLE
Avionics			NLE		
WeapCtrl		NLE			
Refuel	NLE				NLE
Rearm	NLE			NLE	

Table 1: Tasks Interactions Indicated by *NLE* **for** *Non-Local Effect*. **In this paper, NLEs are all mutual exclusion where tasks that interact cannot be performed on the same aircraft at the same time (spatial + temporal MUX). Other NLEs supported include effects like hindering where tasks can be performed together but will slow each other down in some quantified way.**

aircraft is assigned a deadline by which it is to be ready for redeployment. Mission Control is responsible for assigning the deadline and for identifying the areas of the aircraft that need service.

There are five teams on the ground that ready the aircraft for their next mission. Each team is controlled by a coordination decision support agent that uses TÆMS agent technology to reason about what the team should be doing, when, and with which resources. In this scenario the following teams handle aircraft preparation: 1) refuel, 2) rearm (replaces depleted missiles), 3) avionics repair, 4) weapons controls repair, and 5) engines repair. As aircraft land the Mission Control agent notifies the service teams of the aircrafts' service needs and readiness deadlines. The agents then communicate with one another and reason, in a distributed fashion, about how their tasks may interact and how best to select and sequence operations so that the most aircraft can be ready by their respective launch times (if possible – not all problem instances contain fully satisfiable constraints). The agents perform this coordination using a new coordination-key algorithm presented in later sections.

In this scenario the tasks required to repair an individual plane do not need to be performed in any specific sequence, however, there are sets of tasks that cannot be performed simultaneously because they involve the same geographic regions of the aircraft. For instance, the engines cannot be serviced while a plane is rearmed as both of these activities take place on or near the wings. In contrast, avionics can be serviced while an aircraft is rearmed because avionics reside in the cockpit region and the rearming takes place on or about the wings. A full specification of task interactions is shown in Table 1.

There are several characteristics of this problem instance that make it a hard problem:

The situation is dynamic – it is unknown *a priori* in what state the planes will be when they return from their mission. Thus the agents must coordinate and decide which operations to perform in real-time.

Agents must make quantified / value decisions – different tasks have different values and require different amounts of time and labor resources. For instance, it may not be necessary to refuel the aircraft before the next mission but servicing avionics may be critical.

Coordination is dynamic – the operations being performed by the repair teams interact and the occurrence of the interactions are also not known *a priori*. For instance, until an aircraft lands it is not known whether an engine will need servicing at the same time that a refueling crew is attempting to service the aircraft.

Deadlines are present – aircraft have a deadline by which repairs must be completed and different aircraft may have different deadlines. Without deadlines an inefficient algorithm will generally still service all of the aircraft. Deadlines require the agents to reason about end-to-end processes and to coordinate with other agents to optimize their activities. (This type of agent coordination problem is conceptually *dynamic distributed scheduling*.)

Tasks are interdependent – tasks interact in two different ways: 1) over shared resources in a spatial/temporal fashion, 2) multiple tasks must be performed to accomplish a goal, e.g., $Ready(Aircraft_i) = true\ \ iff\ \ AvionicsFixed_i$ $\wedge\ WeaponsControlsFixed_i\ \wedge$ ... (though in TÆMS this generally pertains to degrees of satisfaction rather than a boolean or binary value).

Not always possible to meet all deadlines – not all problem instances are solvable in the sense that in a given scenario, it may be possible that the optimal solution is to miss one aircraft deadline rather than missing many deadlines. When control is distributed, this characteristic can make it particularly difficult to converge on a solution because it is difficult to know whether an optimal result has been achieved (without a complete, global view and a centralized scheduling technology). This characteristic means that it is fairly easy for a coordination algorithm to lead to many planes being partially serviced and none of them actually meeting their deadlines.

This problem instance requires three classes of simulation activities: 1) simulating the outcome of the last mission in terms of aircraft condition, 2) simulating the activities of Mission Control and the initial damage assessment team, 3) simulating the activities of the repair crews. While detailed description is beyond the scope of the paper, from a high level, the aerial battle is simulated using either a problem space generator or a human generator who selects aircraft from a palette and "breaks" the aircraft. The activities of Mission Control and the initial damage assessment team are captured in TÆMS task structures that are produced by the generation tools. In essence, the Mission Control agent "sees" an aircraft for the first time at its specified landing time and at that same time a description of the aircraft's service needs is transmitted to Mission Control in TÆMS format. Mission Control then disseminates the information to the service teams. The activities of the service teams are simulated using the TÆMS agent simulation environment [18]. In this environment the agents, which are distributed on different machines and execute as different processes, communicate and carry out simulated tasks. The simulated tasks, like real tasks, take a specified amount of time to execute and consume resources, e.g., replacing an avionics module of type 1 consumes one type 1 avionics module.

Space precludes a detailed specification of tasks and attributes, however, it is important to note that different tasks require different resources, different amounts of resources, and require different time to perform. For instance, refueling an aircraft that is fully depleted requires more time and consumes more fuel (a resource). Other examples: repairing engines damaged to level 4 (heavily damaged) requires more time than engines that are damaged to level 1 (lightly damaged), rearming four missiles requires more time than rearming two missiles, etc. Similarly, different aircraft consume different resources and not all aircraft need a particular class of service. For instance, the C9 surveillance aircraft does not carry missiles and does not contain a weapons controls module. In contrast, both the A10 and the F16 carry missiles and both have weapons controls modules but the modules for the two aircraft are different and require different amounts of time to service. The teams themselves also maintain different resources, e.g., the refueling team is the only team that

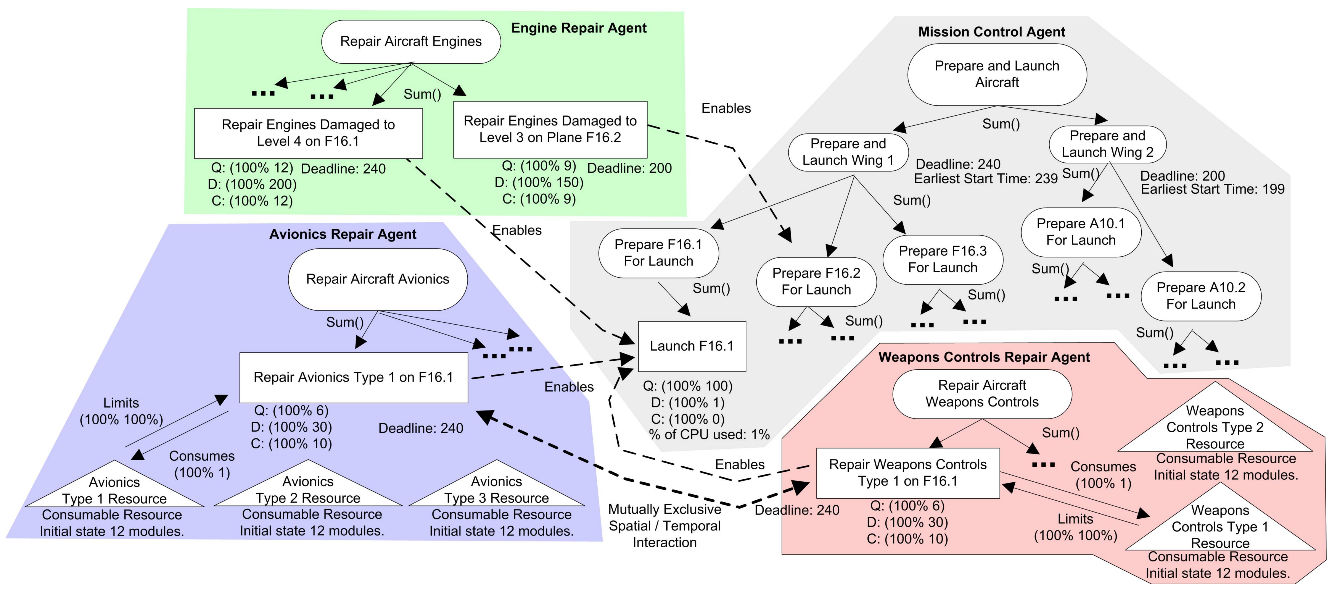


Figure 2: Portions of the TÆMS Task Structures for Mission Control and Three of the Service Team Agents

consumes the fuel resource. However, in the problem instance discussed in this paper the teams do not interact over consumable resources so the coordination problem is one of spatial and temporal task interaction.

It is worth noting that while both DARPA and Honeywell have interest in this particular type of application, the characteristics of the application can be found in other problem domains. The underlying technical problem is to coordinate distributed processes that affect one another when the environment is dynamic and the coordination problem cannot be predicted offline / *a priori* but instead must be solved as it evolves.

3. TÆMS AND TÆMS AGENTS

We use the expression *TÆMS agents* to describe our agent technology because the cornerstone of our approach is a modeling language called TÆMS (Task Analysis Environment Modeling and Simulation) [4]. TÆMS is a way to represent the activities of a problem solving agent – it is notable in that it explicitly represents alternative different ways to carry out tasks, it represents interactions between activities, it specifies resource use properties, and it quantifies all of these via discrete probability distributions in terms of quality, cost, and duration. The end result is a language for representing activities that is expressive and has proven useful for many different domains including the BIG information gathering agent [12], the Intelligent Home project (IHome) [10], the DARPA ANTS real-time agent sensor network for vehicle tracking [7], distributed hospital patient scheduling [3], and others like distributed collaborative design, process control, agents for travel planning, agent diagnosis, and others.

Figure 2 shows portions of TÆMS task structures for Mission Control and three of the service teams. Consider the Mission Control task structure. It is a hierarchical decomposition of a top level goal which is simply to `Prepare and Launch Aircraft`. The top level goal, or task, has two subtasks which are to `Prepare and Launch Wing1` and `Prepare and Launch Wing2`. Each of these tasks are decomposed into subtasks to service a particular aircraft in the given wing, e.g., `Prepare F16.1 For Launch`, and finally into primitive actions. Tasks are represented with oval boxes, primitive actions with rectangles. Note that most of the decompositions are omitted from the figure for clarity. The details are shown for the `Prepare F16.1 For Launch` task – it is decomposed into a single primitive action, `Launch F16.1`, which denotes the time required for Mission Control to launch the aircraft when the plane is ready. The operative word here is *ready*. In order for a given aircraft to be launched on its next mission, it must be serviced. The service activities are not carried out by Mission Control. In the figure, Mission Control's dependence on the activities of the service agents is denoted by the edges leading into `Launch F16.1` from the actions of other agents. These edges, called *enables* in TÆMS, denote that the other agents must successfully perform their tasks before the `Launch F16.1` activity can be carried out by Mission Control. These *enables* are *non-local-effects* (NLEs) and identify points over which the agents must coordinate. The time at which Mission Control can execute `Launch F16.1` is dependent on when the other agents perform their tasks. A different type of NLE exists between the Weapons Controls Repair agent and the Avionics Repair agent – the two F16.1 actions cannot be performed simultaneously and that is another point over which the agents must coordinate. In this problem, this spatial/temporal interaction of the service teams is the coordination problem on which we focus. The former enabling-of-the-launch-task interaction only requires that the service agents notify Mission Control of when they plan to perform their activities because in this application Mission Control sets and maintains deadlines and the other agents negotiate over the temporal/spatial MUX NLEs to satisfy the stated deadlines if possible. Note that within a task structure deadlines and earliest-start-times are inherited (unless those lower in the tree are tighter) so the temporal constraints on `Prepare and Launch Wing1` also apply to `Launch F16.1`. The same deadlines are propagated through the *enables* coordination to the service team agents – note that F16.1's engines must be serviced by 240 also.

Note that all of the primitive actions (leaf nodes) also have Q (quality), C (cost), and D (duration) discrete probability distributions associated with them. For simplicity in this paper we do not use uncertainty and all values will have a density of 100%. Repairing the engines of F16.1 thus takes 200 time units while servicing the engines of F16.2, which are less damaged, requires 150 time units. The two activities produce qualities of 12 and 9 respectively. The `sum()` function under most of the parent tasks is called a *quality-accumulation-function* or *qaf*. It describes how quality (akin to utility) generated at the leaf nodes relates to the performance of

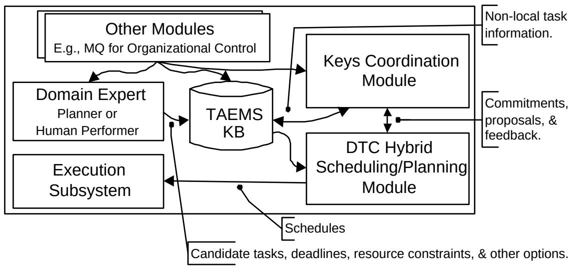

Other Modules
E.g., MQ for Organizational Control
Keys Coordination Module
Domain Expert
Planner or
Human Performer
TAEMS KB
DTC Hybrid Scheduling/Planning Module
Execution Subsystem
Non-local task information.
Commitments, proposals, & feedback.
Schedules
Candidate tasks, deadlines, resource constraints, & other options.

Figure 3: A Single TÆMS-based Agent Ready to Coordinate Its Activities With Other Agents

the parent node. In this case we sum the resultant qualities of the subtasks – other TÆMS functions include min, max, sigmoid, etc. Quality is a deliberately abstract concept into which other attributes may be mapped. In this paper we will assume that quality is a function of the importance of the repair.[2]

In the sample task structure there is also an element of choice – this is a strong part of the TÆMS construct and important for any dynamic environment in which resources or time may be constrained. The `Repair Aircraft Engines` task, for example, has two subtasks joined under the `sum()` qaf. In this case the Engine Repair agent may perform either subtask or it may perform both depending on what activities it has time for and their respective values. The explicit representation of choice – a choice that is quantified by those discrete probability distributions attached to the leaf nodes – is how TÆMS agents make contextually dependent decisions.

By establishing a domain independent language (TÆMS) for representing agent activity, we have been able to design and build a core set of agent construction components and reuse them on a variety of different applications (mentioned above). TÆMS agents are created by bundling our reusable technologies with a domain specific component, generally called a domain problem solver, that is responsible for knowing and encapsulating the details of a particular application domain. For this paper it is sufficient to know that TÆMS agents have components for scheduling and coordination that enable them to 1) reason about what they should be doing and when, 2) reason about the relative value of activities, 3) reason about temporal and resource constraints, and 4) reason about interactions between activities being carried out by different agents. A high-level view of a TÆMS agent is shown in Figure 3; everything except for the domain problem solver is reusable code. Note that each module is a research topic in its own right. The agent scheduler is the Design-to-Criteria [14, 19, 20] scheduler and the coordination module is derived from GPGP [3]. Other modules, e.g., learning, can be added to this architecture in a similar (conceptual) plug and play fashion. In the aircraft service application there are two types of domain problem solvers, those that manage the service teams and the problem solver that handles mission control. The difference is because Mission Control has unique responsibilities (initial assessment, information dissemination, and control over deadline relaxation – not used in the experiments presented here).

4. COORDINATION VIA KEYS

The goals of coordination in this application are: 1) to adapt to a dynamic situation, 2) to maximize the number of planes that are completely repaired by their respective deadlines, 3) to provide mutual access to shared physical resources, 4) achieve global optimization of individual service team (agent) schedules through local mechanisms and peer-to-peer coordination. When examining the coordination problem, it became clear that this application domain has a unique property not generally found in TÆMS agent applications – for agents whose tasks interact, *all* of their tasks will interact. By way of example, all of the engine repair tasks interact with all of the refueling tasks interact with all of the rearming tasks. Similarly for the tasks that pertain to the cockpit. All avionics tasks interact with all weapons controls tasks.

The implications of this property for coordination are that: 1) there is no reason for a service team that operates on the wing region to interact with a team that operates in the cockpit and vice versa[3], 2) agents that operate on the same spatial area (wing or cockpit) must always coordinate their activities. This translates into a discrete partitioning of the agents into *coordination sets*, i.e., $Agents_{wing} = \{Repair\ Engines,\ Refuel,\ Rearm\}$. $Agents_{cockpit} = \{Repair\ Avionics,\ Repair\ Weapons\ Controls\}$, where $Agents_{wing} \cap Agents_{cockpit} = \{\varnothing\}$. Within each coordination set the tasks of the member agents form a fully connected graph via TÆMS *non-local-effects*. This means that for any agent of a given set, e.g., the engine repair agent of $Agent_{wing}$, to schedule a repair task it must dialog with the other agents to ensure that mutual exclusion over the shared resource, e.g., the wing on plane F16.1, is maintained.

This coordination problem could be solved in typical GPGP [4, 3, 11] fashion. However, GPGP operates in a *pairwise* peer-to-peer fashion. For agents in $Agents_{wing}$ this means that coordination could require a significant amount of time to propagate and resolve the interacting constraints and it is unclear given the dynamics of the environment and the speed with which coordination must occur whether convergence on a reasonable, if suboptimal, solution would ever occur.[4] Because of the strong interconnectedness of the tasks and the partitioning of agents into coordination sets, we developed a new algorithm for problem classes of this type.[5]

The algorithm uses a *coordination key* data structure and concepts from token-passing [17, 8] algorithms to coordinate the agents. The general operation of the algorithm is that there is one coordination key per coordination set that is passed from agent to agent in a circular fashion. When an agent is holding the coordination key for its coordination set, it can 1) declare its intended course of action / schedules, 2) evaluate existing proposals from other other agents, 3) confirm or negate proposals of other agents, 4) make its own proposals, or 5) read confirmations or negations of its own proposals by other agents. The coordination key itself is the vehicle by which this information is communicated. Each key contains intended courses of action, proposals, and proposal responses, and this information is modified as the agents circulate the given key. The pseudo-code of the algorithm is shown in Figure 4.

The coordination key algorithm is effective but approximate and heuristic. The crux of the matter is that in order for the agents to coordinate optimally over a single issue, e.g., when agent X should perform task T_1, the key must circulate through the coordination set multiple times. The number of times that each agent must hold the

[2]Beyond the scope of this paper is the use of quality for commitment satisfaction in this application and its role in the control heuristics.

[3]An indirect interaction occurs when the problem instance contains deadlines that cannot be met. In such cases both wing and cockpit agents should forgo work on selected planes in order to avoid having an entire fleet of aircraft that are partially complete, none of which are ready for their next mission. This interaction is dealt with using value for commitment satisfaction and algorithms/experiments pertaining to that topic must be presented separately due to space limitations.

[4]Note that this would also apply to other agent sets if the problem were expanded.

[5]Comparison between the new key algorithm and a pairwise technique is discussed in the conclusion.

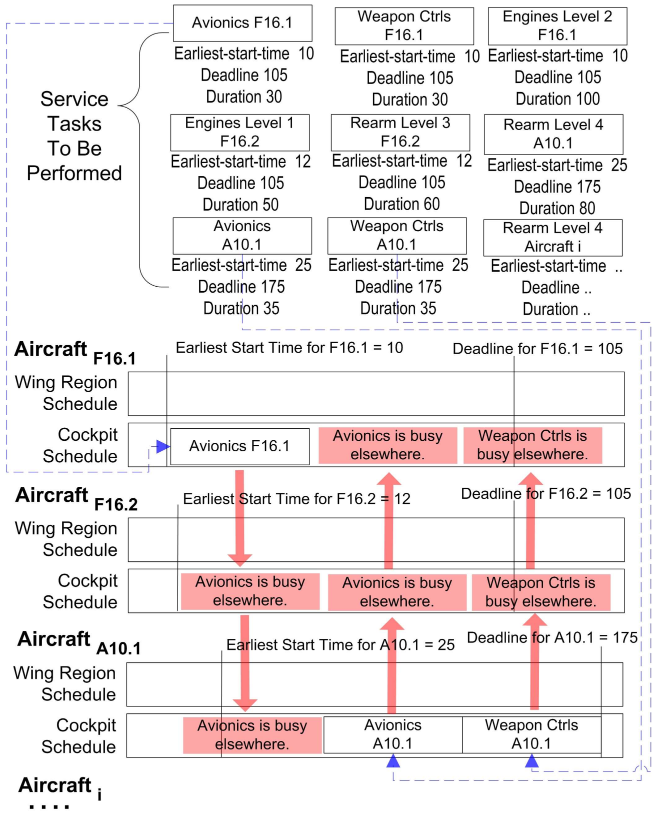
```
If (coordinationKey is not null) and
  (needCoordinate or coordianationKey.othersNeedCoordinate) {
    primarySchedule = evaluate(taems,
                        coordinationKey.getPrimaryDontCommitments());
    if (coordinationKey.getSecondaryDontCommitments()
       interractWith taems.getDeadlineCommitments()) {
          secondarySchedule = evaluate(taems,
                        coordinationKey.getSecondaryDontCommitments());
          if (primarySchedule.quality > secondarySchedule.quality) {
                preferredSchedule = primarySchedule;
            coordinationKey.discardSecondaryDontCommitments();
          } else {
                preferredSchedule = secondarySchedule;
                coordinationKey.replacePrimary-
                      DontCommitmentsWithSecondaryDontcommitments()
          }
    } else {
          preferredSchedule = primarySchedule;
    }
    taems.setSchedule(preferredSchedule);
    violatedDeadlines = taems.getViolatedDeadlines(preferredSchedule);
    newViolatedDeadlines = violatedDeadlines.getNewDeadlines();
    whatifDontCommitments = coordinationKey.getPrimaryDontCommitments();
    whatifDontCommitments.discardInteractions(newViolatedDeadlines);
    whatifSchedule = evaluate(taems, whatifDontCommitments);
    if (whatifSchedule != preferredSchedule)
          coordinationKey.addSecondaryDontCommitments(whatifSchedule);
          whatifViolatedDeadlines = taems.getViolatedDeadlines(whatifSchedule);
          taems.markAsOldDeadlines(whatifViolatedDeadlines)
    }
    oldViolatedDeadlines = violatedDeadlines.getOldDeadlines();
    communicateDeadlineViolation(oldViolatedDeadlines);
```

Figure 4: Pseudo-code of an Individual Agent's Coordination Reasoning

key is dependent on the changes made during each iteration. In the worst case each agent will have to re-sequence each of its n activities once for every change that is made, but these changes propagate to the other agents so the circulation-to-convergence factor is $O(x^n)$ rather than $O(n^x)$ (where x is a constant). The coordination key algorithm above multiplexes changes so that in a given pass through a coordination set multiple changes are considered by the agents at once.

We hypothesized that in some problem instances the algorithm would fail to find an optimal solution but that in most problem instances it would perform well. To test this hypothesis we created a centralized global scheduler that creates schedules for all of the agent teams via exhaustive search. The centralized scheduling problem is exponential, however, for instances having less than 11 total repairs the exhaustive scheduler is responsive enough for experimentation.[6] Because the problem instance presented here uses a subset of TÆMS features, the centralized scheduler is designed to solve a representation of exactly the subset needed, i.e., it does not perform detailed TÆMS reasoning but instead maintains the required constraints (e.g., deadlines, earliest start times, service teams can only service one aircraft at a time, and only one service team can work in a cockpit or the wing region at a given point in time). The centralized scheduler algorithm is outlined in Figure 5. The function of the centralized scheduler is twofold. First, it determines the minimum number of aircraft deadlines that will be missed by an optimal solution. In some cases all deadlines can be met and in others aircraft deadlines represent unsatisfiable constraints. The second role of the centralized scheduler is to determine the relative size of the different solution spaces. For instance, for a given problem

[6]The centralized scheduler requires on the order of 10 minutes to schedule 11 repairs on a dual-processor Xenon 2Ghz linux workstation. A problem instance of that size will generate on or about 241,920 schedules some subset of which are unique.



Constraints to maintain: 1) Service team on a single aircraft at a given instant in time. 2) One service team in each region at a given instant in time. 3) Earliest start times which denote when the aircraft lands. 4) Deadlines which denote when the aircraft is due for its next mission.

Figure 5: The Centralized Exhaustive Scheduling Oracle Has An Omnipotent View – Figure Shows One Scheduling Instance

there may be zero solutions that don't miss any deadlines, X (optimal) solutions that miss one aircraft deadline, Y solutions that miss two aircraft deadlines, Z solutions that miss three aircraft deadlines, etc. By tabulating this information we can determine a percentile ranking for the solutions produced by the distributed coordination key algorithm. The centralized scheduler does not compete with the distributed coordination key algorithm on a completely level playing field. The centralized scheduler sees all the repairs that will be needed for all planes on a given problem instance at time 0. The agents in the distributed system only see repairs as the aircraft land. Thus, for the instance shown in Figure 5, the service team agents will not see aircraft A10.1 until time 25 (when it lands). At this time they may be committed to a suboptimal course of action that the centralized omnipotent scheduler will avoid because it can see A10.1's repairs at time 0 along with all of the other repairs that will need to be scheduled. This difference is due to a need to keep the centralized scheduler development costs down and has its roots in design/implementation issues with the simulation environment. A related bias in favor of the centralized scheduler is that the distributed coordination mechanisms operate in the same simulated clock as the repairs themselves. This enables the simulation environment to control and measure coordination costs but causes a skew in terms of the apparent cost of coordination relative to domain tasks, e.g., in some cases the ten clicks (about 5 seconds in wall clock time) that the agents require to coordinate will take as much simulation time as it takes the service teams to rearm one missile on an aircraft. The skew is of primary relevance when comparing the distributed

Expmnt Class	Num Trials	Mean # Solutions Possible Missing X Aircraft Deadlines X=0	X=1	X=2	X=3	X=4	X=5	X=6	Characteristics of Solution Generated by Coordination Keys Mean	%-tile	Median %-tile	StdDev of %-tile	%-tile Same	%-tile Better	%-tile Worse
A	32	.31	1.09	.75	.31	.13	0	0	1.13	1.0	1.0	0	.80	0	.20
B	32	.31	2	3.1	3.6	2.9	0	0	1.5	.98	1.0	.12	.58	.02	.41
C	32	0	3.2	13	24.1	16.3	2.53	.4	2.1	.97	1.0	.08	.38	.03	.62
D	28	0	3.1	16.8	33.0	49.6	36.3	2.7	2.4	.98	1.0	.04	.26	.02	.73

Table 2: Results Comparing Coordination Keys to Exhaustive and Optimal Centralized Schedule Generation

algorithm to the centralized scheduler and is less of an issue when comparing different distributed algorithms.

Table 2 presents the results of comparing the coordination key algorithm to the optimal and exhaustive centralized scheduler. Each row is the statistical aggregation of one set of trials where each set of trials is drawn from one difficulty class. The rows lower in the table represent increasingly more difficult problem instances – aircraft having more repairs and tighter deadlines relative to their landing times and the time required for their repairs [7]. All rows except for the last represent 32 random trials. Row D contains 28 because of the occasional exception thrown by the exhaustive scheduler caused by running out of RAM. As the difficulty increases, note that the density of the solution space increases and shifts right. This is represented by the columns *X=0, X=1, ...,* which contain the mean number of solutions produced by the oracle that miss 0 deadlines, 1 deadline, etc., respectively. As the problem instances get harder more aircraft are likely to miss deadlines. Note that the coordination key algorithm generally performs well for all of the tested conditions. The *Mean* value denotes the average number of aircraft deadlines missed during a batch of trials. The more descriptive statistics are those about the percentile ranking of the solutions generated by coordination keys. This is because how well the keys algorithm performs is determined not by the absolute number of missed deadlines (the average of which is presented in the *mean* column) but instead by the solutions possible for a given trial. For instance, in some trials the best solution possible may miss two deadlines. As the difficulty increases the *mean* value for the keys algorithm increases because there are more instances where the optimal solution is to miss one deadline, or two deadlines, etc. Looking at the percentiles, in experiment class A the keys algorithm performed in the 100th percentile, in experiment class B the 98th percentile, in experiment class C the 97th percentile, and in class D (the most difficult class), the 98th percentile. The percentile rating is computed as follows:

- The centralized scheduler generates all of the unique schedules that exist for a given individual trial.
- These schedules are binned according to the number of deadlines missed, e.g., in X of the schedules 0 aircraft miss a deadline, in Y of the schedules 1 aircraft misses a deadline, in Z of the schedules 2 of the aircraft miss a deadline, etc. Think of the centralized scheduler as producing a histogram of possible solutions where solutions are binned by the number of deadlines missed.
- Let $CKDLM_i$ be the number of aircraft deadlines missed by the coordination key algorithm in trial i.
- Let Bin_CK_i denote the histogram bin in which $CKDLM_i$ falls (the bin that pertains to $CKDLM_i$ missed deadlines).
- Let $Density\_at\_or\_above_i$ be the $\sum$ of the densities of solutions that are in bins > or $=$ to Bin_CK_i. Bins $> Bin\_CK_i$ represent solutions that are *worse* because they entail missing more deadlines.
- Let $Percentile\_Ranking_i$ = $Density\_at\_or\_above_i / TN_i *$ 100, where TN_i is the total number of solutions generated by the centralized scheduler for trail i. $Percentile\_Ranking_i$ is the percentile ranking for the coordination key algorithm for trial i of the set of 32.
- Let $Overall\_Percentile\_Ranking =$ $(\sum_{i=1}^{32} Percentile\_Ranking_i)/32$ be the overall percentile ranking for one batch of 32 trials.

In all cases the median percentile is 100% and the standard deviation is low. Because there are generally multiple solutions that perform as well as the solutions actually generated by the coordination keys, its percentile is broken down in the last three columns of Table 2. The column marked *%-tile Same* indicates the mean % of possible solutions that miss exactly as many deadlines as the keys algorithm did. *%-tile Better* indicates the number that performed *strictly better* (missing fewer aircraft deadlines) and *%-tile Worse* indicate the number that performed strictly worse. Note that as the problem space gets harder the number of solutions possible that are worse than those found by the keys algorithm increases. At the same time the band of solutions as good as those generated by keys narrows, as does the band of solutions that are strictly better than those found by the keys algorithm.

While the data suggests that the algorithm performs well on average, there are circumstances where the algorithm performs less well. We examined several such instances in detail and while we have intuitions about when the algorithm will perform in a suboptimal fashion, the experiments in which performance is suboptimal pertain to a more basic issue. To illustrate let us assume a three-aircraft problem instance with the following characteristics:

- Aircraft F16 arrives at time 15 with a deadline or take-off time of 400 and requires repair of engines damaged to level 2 (the duration of this repair is 100).
- Aircraft A10 arrives at time 18 with a deadline of 450 and requires complete refueling (the duration of this task is 100).
- Aircraft C9 arrives at time 24 with a deadline of 240 and requires repair of engines damaged to level 2 (the duration of this repair is 100) and refueling of a quarter tank (duration of this tank 25).

The F16 lands at time 15 and the engine service team obtains the coordination key and schedules the engine repair of the F16 to run from time 17 to 117. The A10 lands at time 18 and at time 19 the refuel team gets the coordination key and schedules refueling of the A10 to last from 19 to 119. When the C9 lands at time 24 the engine service team is thus occupied with the F16 until time 117 and the refueling team is occupied with the A10 until time 119. To respond to

[7]The seven trial parameters are: (1) land time, (2) takeoff time deadlines, (3) level of avionics damage, (4) level of weapons control damage, (5) level of engines damage, (6) level of rearm damage, and (7) refuel level.

the C9's landing and repair needs, the engine service team obtains the coordination key at time 25 and schedules C9's repair to run from time 117 to time 217. At a subsequent time-step, the refueling team attempts to schedule C9's refueling, however, because both refueling and engine repair are mutually exclusive tasks, the earliest time the refueling team can schedule the C9 is at time 217. This means it is impossible to service the C9 by its deadline (take-off time) of 240. In response to this pending failure, the refuel service team attempts to negotiate with the engine service team via the coordination key to obtain a wing access slot between 119 and 217. However, the engine service team needs that time slot to complete its portion of the C9's engine repairs on time. The end result is that the C9's deadline cannot be met. For this same problem instance, however, the centralized scheduler was able to produce a solution in which all of the deadlines are met.

The underlying issue is that service activities are not interruptible in this problem instance – otherwise repair teams could run from aircraft to aircraft and the optimization problem would be much simpler. If activities were interruptible, when the C9 first landed either the engine service team or the refuel service team could disengage from their respective current activities (servicing the F16 or the A10) and attend to the C9, which is the aircraft with the tightest deadline. The reason the centralized scheduler is able to produce a better solution in this problem instance – a solution which eludes the distributed coordination approach – is that the centralized oracle sees all of the repair tasks *a priori*. It thus considers the possibility of not servicing the F16 or A10 immediately upon arrival so that the C9 can be serviced by engines or refueling immediately upon its arrival and all deadlines can be met.

This particular performance issue derives from the somewhat imbalanced playing field (discussed earlier) between the distributed algorithm and the centralized oracle. Interestingly, we can hypothesize two instances where the distributed algorithm will fail to perform well, even on a level playing field, but such instances occur infrequently in randomly generated problem instances – even those with tight deadline constraints and numerous repairs per aircraft.[8]

One instance where the the coordination key algorithm will perform less well entails semi-independent coordination problems that occur simultaneously in the coordination set of more than two agents. Imagine a coordination set of the rearm, refuel, and engine repair agents. Let the key pass from agent to agent in the following order: rearm to refuel to engine (then the cycle repeats). Now, let us assume that at time t the rearm agent needs a time slot that is held by the engine agent, and that refuel needs a time slot that is held by the rearm agent. The implications are that multiple unrelated proposals must reside on one key for part of the coordination set traversal, i.e., the proposal from rearm to engine and the proposal from refuel to rearm both reside on the key during the refuel to engine to rearm circuit. The key algorithm is designed with the assumption that, in general, multiple proposals will pertain to a single (sometimes multi-step) coordination process. Therefore, when the engine agent receives the coordination key it either accepts or rejects the set of current proposals (from the rearm and refuel agents) *en masse* even though it may only be affected by the rearm agent's proposal. In this case, when the set of proposals arrives and the engine agent determines that it cannot satisfy the rearm agent's request, it rejects the proposals *en masse* and the proposal from refuel to rearm is never evaluated by the rearm agent. This may result in a missed opportunity for the refuel agent. The shortcoming described here can be fixed by making the agents more selective in proposal rejection.

[8]If the repairs are spread over a large number of aircraft there is little spatial resource contention and service teams can basically function in parallel.

Another instance where the coordination key algorithm may perform less well is when a long chain of multi-step inter-locking resource releases are required. The factor at work is the algorithm's approximate limited-cycle-to-action model. However, as noted, neither class of problems occur frequently with random instances. We are currently exploring creating a generator and experiments to test performance under these circumstances.

5. RELATED WORK

The TRACE system [5] also deals with distributed task performance though the focus of TRACE is on task allocation to agent organizations. Like our problem space, TRACE tasks have deadlines and required durations. The important difference is that in TRACE the tasks are independent – they do not interact over shared resources at run or performance time nor are sets of them required to solve some larger objective (e.g., in our application there is conceptually an and function over the set of service tasks required to prepare the plane for its next launch).

The intelligent home, or IHome [10] project, also had resource coordination issues to resolve. The *SHARP* protocol used in IHome was implemented in both centralized and distributed versions. The distributed version is the most relevant for comparison though it differs from the key algorithm in several ways. First, *SHARP* relies on client assigned priorities to determine which agent should have access to a given resource if there is contention. Second, in the IHome activities are fixed temporally, e.g., the human user takes his/her shower at a particular time of day so if there is a conflict, the priority measure alone determines proper resolution. *SHARP* also does not partition the agents so agents that did not consume resource X would still coordinate over the usage of resource X. *SHARP* also uses a general broadcast model which means interacting issues can be broadcast concurrently – we believe this would not work well for conflict resolution in a domain without hard-set priority measures.

Work in economic agent systems frequently entails protocol development, e.g., [9, 2]. The issues addressed by such protocols are generally a different class. Instead of the global optimization of interacting tasks with resource and temporal constraints, such work generally relates to fairness, bidding strategies within the problem space, and obtaining equilibrium. Such work also generally entails a centralization mechanism, e.g., an auctioneer, not present in the distributed dynamic coordination of this paper. Note that combinatorial auctions [15] exhibit intractability in much the same way as distributed scheduling-esque coordination work like that presented here do.

Peer-to-peer auctions [13] in which bidding is distributed also exhibit propagation-to-convergence properties. However, because the attributes being modified are the bid and ask prices, and these are not fixed temporally the way repair tasks are situated in our problem instance, the parallels are generally surface level.

Other TÆMS coordination technologies exist and have been used on related problems. In the distributed sensor network [7] agents coordinate to track vehicles through a sensor grid. This problem and the control regime differ because the problem focuses on periodic tasks. Another example is distributed hospital patient scheduling [3]. The coordination mechanisms used in that application are the traditional GPGP pairwise mechanisms that we believe would not serve well in the aircraft service problem space due to the higher degree of interdependence (discussion continues in the conclusion).

With respect to the community at large, there are some similarities between our work and that dealing with teamwork [16] or joint problem solving formalisms [6]. Our work differs in its distributed dynamic scheduling focus and its implicit, rather than explicit, use of joint goal constructs. (More details are in [11].)

The coordination key algorithm has some concepts in parallel with token based networking [17, 8]. One parallel is that the *coordination set* of agents form a virtual ring around which the key is passed. The key, like the token in networking, gives the agent holding the key the ability to enter its conceptual critical section. In networking, the critical section is actually the network bandwidth – in our coordination algorithm the critical section is the ability to propose changes and the ability to respond to other proposals. Our work differs in that the key represents the ability to make high level changes that impact the other nodes and that the key itself carries proposals, responses to proposals, and other coordination data.

6. LIMITATIONS AND FUTURE WORK

In the future, we would like to compare the coordination key algorithm to a distributed pairwise (classic GPGP style) algorithm. We believe the key-based approach will perform better but this is only conjecture at this point. Because the character of the spatial interactions in this problem differs from that typically modeled in TÆMS the standard GPGP coordination techniques could not be employed without modifications to TÆMS. Due to time and resource constraints, and a desire to compare distributed coordination to a centralized optimal oracle, resources were directed in that fashion.

Hereto unmentioned is the possibility of creating an efficient optimal centralized scheduler for the service teams. As presented here the task space appears sufficiently constrained to lend itself to centralized approaches that do not require exhaustive search. This is partly an artifact of the task space as framed for this application – we are using a small subset of TÆMS features and the non-local-effects (NLEs or task interactions) presented in this paper all involve mutual exclusion. In the general case, task interactions may impact each other's quality, cost, and durations (not just dictate mutual exclusion) and the element of choice is larger than presented here – these characteristics combined with differing deadlines often thwart typical non-exhaustive centralized scheduling methodologies. More importantly, however, is the motivation for distribution. Distribution enables incremental addition of repair teams, gives each team local autonomy (if the simulated teams were human, they could exercise their own judgment and the TÆMS technologies would coordinate with the human's choices when the coordination recommendation is over-ridden), removes a central point of failure, and removes the issue of computational or communication overhead that occurs with one centralized scheduling node. In the broader sense, centralization is often not possible or not desirable due to privacy concerns, the need to avoid a central point of failure, scalability issues, or the potential processing delay. In many instances, it is also not required – consider the discrete spaces represented by the different coordination sets in this application but imagine a network of 100,000 agents – not all of these would need to interact, coordinate, or even be aware of one another.

7. REFERENCES

[1] F.M. Brazier, C.M. Jonker, and J. Treur. Formalization of a cooperation model based on joint intentions. In *Intelligent Agents II*, Springer Verlag, 1997.

[2] E. David, R. Azoulay-Schwartz, and S. Kraus. Protocols and strategies for automated multi-attribte auctions. In *Proc. of the 1st Intl. Conf. on Autonomous Agents and Multi-Agent Systems*, 2002.

[3] K. Decker and J. Li. Coordinated hospital patient scheduling. In *Proc. of the 3rd Intl. Conf. on Multi-Agent Systems*, pages 104–111, 1998.

[4] K. Decker. *Environment Centered Analysis and Design of Coordination Mechanisms*. PhD thesis, University of Massachusetts, 1995.

[5] S. Fatima and M. Wooldridge. Adaptive task and resource allocation in multi-agent systems. In *Proc. of the 5th Intl. Conf. on Autonomous Agents*, 2001.

[6] B. Grosz and S. Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 86:269–357, 1996.

[7] B. Horling et al. Distributed sensor network for real-time tracking. In *Proc. of the 5th Intl. Conf. on Autonomous Agent*, 2001.

[8] IEEE. *802.5: Token Ring Access Method*. IEEE, New York, NY, 1985.

[9] T. Ito, M. Yokoo, and S. Matsubara. Designing an auction protocol under asymmetric information on nature's selection. In *Proc. of the 1st Intl. Conf. on Autonomous Agents and Multi-Agent Systems*, 2002.

[10] V. Lesser et al. A Multi-Agent System for Intelligent Environment Control. In *Proc. of the 3rd Intl. Conf. on Autonomous Agents*, 1999.

[11] V. Lesser, K. Decker, T. Wagner, et al. Evolution of the GPGP Domain-Independent Coordination Framework. UMASS CS Tech. Report TR-98-05. To appear in the Journal of AAMAS, 2003.

[12] V. Lesser et al. BIG: An agent for resource-bounded information gathering and decision making. *Artificial Intelligence*, 118(1-2):197–244, May 2000. Elsevier Science Publishing.

[13] E. Ogston and S. Vassiliadis. A peer-to-peer agent auction. In *Proc. of the 1st Intl. Conf. on Autonomous Agents and Multi-Agent Systems*, 2002.

[14] A. Raja, V. Lesser, and T. Wagner. Toward Robust Agent Control in Open Environments. In *Proc. of the 4th Intl. Conf. on Autonomous Agents*, 2000.

[15] T. Sandholm, S. Suri, A. Gilpin, and D. Levine. Winner determination in combinatorial auction generalizations. In *Proc. of the 1st Intl. Conf. on Autonomous Agents and Multi-Agent Systems*, 2002.

[16] M. Tambe and W. Zhang. Towards Flexible Teamwork in Persistent Teams. In *Proc. of the 3rd Intl. Conf. on Multi-Agent Systems*, 1998.

[17] A.S. Tanenbaum. *Computer Networks*. Prentice Hall, New Jersey, 1996.

[18] R. Vincent, B. Horling, and V. Lesser. An agent infrastructure to evaluate multi-agent systems: The java agent framework and multi-agent system simulator. In *Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems*, pages 102–127. Springer, 2001.

[19] T. Wagner, A. Garvey, and V. Lesser. Criteria-Directed Heuristic Task Scheduling. *Intl. Journal of Approximate Reasoning, Special Issue on Scheduling*, 19(1-2):91–118, 1998. Version also avail. as UMASS CS Tech. Report TR-97-59.

[20] T. Wagner and V. Lesser. Design-to-Criteria Scheduling: Real-Time Agent Control. In *Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems*, LNCS. Springer-Verlag, 2001. Version avail. as UMASS CS Tech. Report TR-99-58.

[21] N.R. Jennings. Specification and Implementation of a Belief-Desire-Joint-Intention Architecture for Collaborative Problem Solving In *International Journal of Intelligent and Cooperative Information Systems*, Vol 2, No 3, 1993, pages 289-318.