, , 1–27 ()
© Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

Coordinating Mutually Exclusive Resources using GPGP

KEITH DECKER AND JINJIANG LI decker@cis.udel.edu
Department of Computer and Information Sciences, University of Delaware

Received March 31, 1999; Revised September 13, 1999

Editor: Yves Demazeau

Abstract. Hospital Patient Scheduling is an inherently distributed problem because of the way real hospitals are organized. As medical procedures have become more complex, and their associated tests and treatments have become interrelated, the current ad hoc patient scheduling solutions have been observed to break down. We propose a multi-agent solution using the Generalized Partial Global Planning (GPGP) approach that preserves the existing human organization and authority structures, while providing better system-level performance (increased hospital unit throughput and decreased patient stay time). To do this, we extend GPGP with a new coordination mechanism to handle mutually exclusive resource relationships. Like the other GPGP mechanisms, the new mechanism can be applied to any problem with the appropriate resource relationship. We evaluate this new mechanism in the context of the hospital patient scheduling problem, and examine the effect of increasing interrelations between tasks performed by different hospital units.

Keywords: Multi-agent systems, coordination, distributed scheduling, applications, organizational modeling and redesign

1. Introduction

Generalized Partial Global Planning (GPGP) is a task environment centered approach to coordination [10, 5]. The basic idea is that each agent constructs its own local view of the structure and relationships of its intended tasks. This view may then be augmented by information from other agents, and it may change in other ways dynamically over time. The GPGP approach uses a set of individual *coordination mechanisms* to help to construct these partial views, and to recognize and respond to particular task structure relationships by making commitments to other agents. These commitments result in more coherent, coordinated behavior [3, 2, 16]. No one coordination algorithm will be appropriate for all task environments, but by selecting from a set of possible coordination mechanisms we can create a wide set of different coordination responses.

We have demonstrated the usefulness of the GPGP approach using five coordination mechanisms drawn from Durfee's original PGP work [12] in several domains—distributed vehicle monitoring [7], distributed data processing [25], and in randomly generated problems [10]. This paper contains a study of GPGP as applied to the hospital patient scheduling domain. The domain is quite interesting because it is inherently distributed due to the existing human authority structure of a hospital.

Although we can apply the five existing GPGP coordination mechanisms to the hospital patient scheduling problem directly, the hospital patient scheduling problem contains a task relationship that is not handled by any of the existing coordination mechanisms, namely, that the hospital patient is a mutually exclusive resource. To address this new relationship, first we must define it formally as a resource constraint in our task structure representation language (TÆMS [9, 32]) and then we define a new GPGP coordination mechanism to handle the new task relationship. We show how the existing practices can be captured using GPGP and *removing* some of the mechanisms. GPGP *with* our new resource-constraint mechanism outperforms existing practice (as described by Ow et al. [23]) in two ways: increased patient throughput, and decreased patient stay time. Historically, hospital procedures were not as mutually interrelated as they are becoming today. We also demonstrate that the existing practice produces similar results to our new mechanism when few inter-unit task relationships exist.

The paper is arranged as follows. First we discuss the hospital patient scheduling problem, and how we represent mutual exclusion resource problems in general using the TÆMS modeling language. Next we briefly discuss GPGP, and focus on the new resource constraint coordination mechanism. Finally, we present our experimental results, and indicate how this work is being extended to broader resource models.

2. The Hospital Scheduling Problem

Our model is drawn from a case study of an actual hospital [23, 18]:

> *Patients in General Hospital reside in* units *that are organized by branches of medicine, such as orthopedics or neurosurgery. Each day, physicians request certain tests and/or therapy to be performed as a part of the diagnosis and treatment of a patient. [. . .] Tests are performed by separate, independent, and distally located* ancillary departments *in the hospital. The radiology department, for example, provides X-ray services and may receive requests from a number of different units in the hospital.*

Furthermore, each test may interact with other tests in relationships such as enables, requires−delay (a slight variation on enables where the second task must be both after and delayed), and inhibits (a negative variation of the soft facilitates relationship where the performance of some test within some timeframe invalidates the results of another). These task relationships indicate when the execution of one task changes the characteristics (here, primarily duration) of another task [9].

From this view of task structure, the hospital scheduling problem has these peculiar features:

- Tasks have no redundancy. Each test can only be done by a single ancillary department.
- The quality accumulation functions of non-executable tasks are always *min*. This is because here all the tests need to be done (TÆMS can however represent a wide range of quality accumulation functions).

- Unlike our previous domain models, here the final quality is not important. A test is either completed, or it is not.
- Different agents (representing different nursing or ancillary units) may use different utility functions when making local schedules. For example, the unit nurses may try to minimize the patient's stay time, while the ancillary staff tries to maximize equipment use and/or minimize setup times.
- Although TÆMS and GPGP handle them, there are no deadlines in this domain model.

Examining the hospital's current coordination structure is enlightening because it shows a mismatch between the structure and the current hospital environment. Although the modern hospital task environment is quite complex and interrelated (see Figure 1), the coordination structure actually used by the hospital assumes that there are no interrelationships! Each ancillary acts independently, without communication with either nursing units or other ancillaries (except for the initial patient order from the nursing unit). The current practice observed in the hospital is for the nursing units to notify ancillaries of the requested tests and treatments, and for all ancillaries to schedule independently, sending an orderly to collect a patient whenever the scheduled time arrives. This somewhat surprising description has been echoed by hospital administrators in several recent discussions as well. Of course, with no inter-ancillary coordination, there is no way to know if the patient will actually be there—they might be off having a different test performed. Unit nurses try to make sure the proper prerequisite tests (represented here by enablement constraints) are done first. While this structure seems sorely lacking when compared to the current environment, it may historically have been a reasonable, low overhead arrangement. It may be that in the past doctors ordered fewer tests on less complex ancillary equipment with fewer or no interfering relationships between these technologies. Thus the current practice was adapted for a different task environment than today. In Section 6, we demonstrate that a simple algorithm, not much different from the current hospital scheduling practice, is not significantly different from our new algorithm when ancillary interrelationships are low or non-existent.

3. Mutually Exclusive Resource Modeling Using TÆMS

TÆMS task structures are abstraction hierarchies whose leaves are instantiated basic actions or "executable methods". At a basic level this is similar to HTN (Hierarchical Task Network) or TCA (Task Control Architecture) approaches to action representation[13, 28]. Additionally, TÆMS allows the specification of dynamically changing and uncertain task characteristics that effect an agent's preferences (utility) for some state of the world, including tasks with hard or soft deadlines. A TÆMS specification also indicates relationships between local and non-local tasks or resources that effect these agent preference characteristics. Thus it extends HTN ideas toward specifying "worth-oriented" domains [26]. Recent extensions to TÆMS

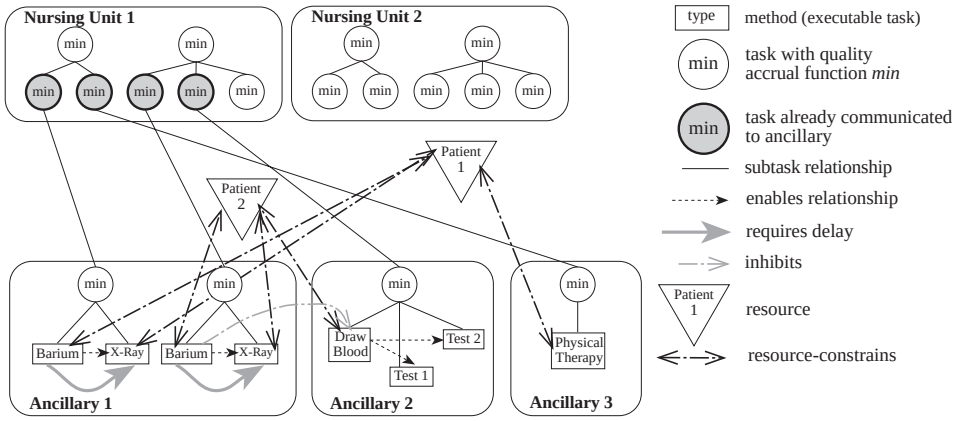

Figure 1. High-level, objective task structure and subjective agent views for a typical hospital patient scheduling episode.

have included named provision relationships and multiple outcome specifications [34, 32].

TÆMS can be used both as a subjective internal representation for agent reasoning (see Section 4), or as an objective modeling language for formally specifying a problem. Here we will present just enough of the TÆMS modeling language to describe our model of mutually exclusive (mutex) resource problems. A complete specifications of the TÆMS modeling language can be found in [5].

In utility theory, agents have preferences over possible final states (action or plan outcomes), and preference-relevant features of an outcome are called *attributes*. A substantial body of work exists on relating attribute values to overall utilities [33]. At its core, TÆMS is about specifying these attributes and the processes by which they change—what we call a model of the task environment. In this paper we will use only two attributes, *quality* and *duration*. Furthermore, in Hospital Patient Scheduling, *quality* is limited to 0 or 1, although our TÆMS statement of the problem is actually much more general.

Actions. A TÆMS *action* (or *executable method*) represents the smallest unit of analysis. $\mathbf{d}_0(M)$ is the *initial duration* of action M, and $\mathbf{q}_0(M)$ is the *initial maximum quality* of action M. $\mathbf{d}(M,t)$ is the *current duration*, and $\mathbf{q}(M,t)$ is the *current maximum quality* of action M at time t. $Q(M,t)$ is the *current quality* of action M. $Q(M,t) = 0$ at times t before the execution of M. If an agent begins to execute M at time t (written $\text{Start}(M)$) and continues until time $t+\mathbf{d}(M,t)$ (written $\text{Finish}(M)$), then $Q(M,\text{Finish}(M)) = \mathbf{q}(M,\text{Finish}(M)) = \mathbf{q}(M,\text{Start}(M))$ (i.e. the current actual quality becomes the maximum possible quality). For the purposes of evaluation in this paper, the amount of work done on an action M here is simply $\text{Work}(M) = \text{Finish}(M) - \text{Start}(M)$. If there were no interrelationships (*non-local effects*, NLEs) between M and anything else, then $\mathbf{q}(M,t) = \mathbf{q}_0(M)$ and $\mathbf{d}(M,t) = \mathbf{d}_0(M)$. The execution of other actions and tasks effect an action

precisely by changing the current duration and current maximum quality of the action (that is, $\mathbf{d}(M,t)$ and $\mathbf{q}(M,t)$, as specified below). For the purposes of this paper, we will also assume that $Q(M,t) = 0$ for $Start(M) \leq t < Finish(M)$; other definitions of Q are possible to represent anytime algorithms, etc. Action preemption and resumption may also be modeled by extending these simple definitions [5].

Tasks. A TÆMS *task* (or *subtask*) represents a set of related subtasks or actions, joined by a common *quality accumulation function*. For example, in an AND/OR tree, an AND task indicates that all subtasks must be accomplished to accomplish the task, while an OR task indicates that only one subtask needs to be accomplished. Since TÆMS is about worth-oriented environment modeling, we use continuous rather than logical quality accumulation functions (for example min instead of AND, max instead of OR[1]). Given a subtask relationship $\mathsf{subtask}(T_1, \mathbf{T})$ where $\mathbf{T}$ is the set of all direct subtasks or actions of T_1, then if T_1 is an AND task we may recursively define $Q(T_1,t) = Q_{\min}(T_1,t) = \min_{T \in \mathbf{T}} Q(T,t)$. For the purposes of evaluation, the amount of work done on a task is the sum of all the work done on its subtasks, and the finish time of a task is the latest (max) finish time of any subtask.

Non-local Effects (NLEs). Any TÆMS action/method, or a task T containing such a method, may potentially affect some other method M through a *non-local effect* e. We write this relation (a labeled arc in the task structure graph) as $\mathsf{nle}(T, M, e, p_1, p_2, \ldots)$, where the p's are parameters specific to a class of effects. For this paper, there are three possible outcomes of the application of a non-local effect on M under our model: $\mathbf{d}(M,t)$ (current duration) is changed, $\mathbf{q}(M,t)$ (current maximum quality) is changed, or both. An effect class e is thus a function $e(T, M, t, d, q, p_1, p_2, \ldots) : [\text{task} \times \text{method} \times \text{time} \times \text{duration} \times \text{quality} \times \text{parameter } 1 \times \text{parameter } 2 \times \ldots] \mapsto [\text{duration} \times \text{quality}]$. For the purposes of this paper, we will ignore the details regarding *where* information is available, i.e. non-local effects that depend on the transmission of information. Our model will use three NLEs: enables, facilitates, and mutex.

Enables. If task T_a enables action M, then the maximum quality $\mathbf{q}(M,t) = 0$ until T_a is "completed", at which time the current maximum quality will change to the initial maximum quality $\mathbf{q}(M,t) = \mathbf{q}_0(M)$. Another way to view this effect is that it changes the "earliest start time" of *enabled* method, because a rational scheduler will not execute the method before it is enabled.

$$\mathsf{enables}(T_a, M, t, d, q, \theta) = \begin{cases} [d, 0] & t < \Theta(T_a, \theta) \\ [d, \mathbf{q}_0(M)] & t \geq \Theta(T_a, \theta) \end{cases} \tag{1}$$

The term $\Theta(T_a, \theta)$ computes the earliest time at which task T_a reaches quality θ.

Facilitates. Computationally, facilitation occurs when information from one task, often in the form of constraints, is provided that either reduces or changes the search space to make some other task easier to solve. A simple to understand example of this relationship in computation is the relationship between sorting and searching. It is faster to retrieve an item from a sorted data structure, but sorting is not *necessary* for retrieval. Hence the sorting task facilitates the retrieval task.

In our framework, one task may provide results to another task that facilitates the second task by decreasing the duration or increasing the quality of its partial result. Therefore the facilitates effect has two constant parameters (called *power* parameters) $0 \leq \phi_d \leq 1$ and $0 \leq \phi_q \leq 1$, that indicate the effect on duration and quality, respectively. The effect varies not only through the power parameters, but also through the quality of the *facilitating* task available when work on the *facilitated* task starts (the ratio R, defined below).

$$\begin{aligned} R(T_a, s) &= \frac{\mathrm{Q}_{\mathrm{avail}}(T_a, s)}{\mathbf{q}(T_a, s)} \\ \mathsf{facilitates}(T_a, M, t, d, q, \phi_d, \phi_q) &= [d(1 - \phi_d R(T_a, \mathrm{Start}(M))), \\ &\quad q(1 + \phi_q R(T_a, \mathrm{Start}(M)))] \end{aligned} \tag{2}$$

So if T_a is completed with maximal quality, and the result is received before M is started, then the duration $\mathbf{d}(M, t)$ will be decreased by a percentage equal to the duration power ϕ_d of the facilitates effect. The second clause of the definition indicates that communication after the start of processing has no effect. In this paper we will only use the duration effect power ϕ_d. Negative values for power parameters produce "hindering" or "inhibition" effects.

Mutex. We first represent mutually exclusive access to a resource R by a set of actions $\mathbf{M}$ as a set of pairwise mutex NLEs between all the elements of $\mathbf{M}$. In Section 8 we extend this model to include *explicit* resources, and more general resource limitations than mutual exclusion.

$$\mathsf{mutex}(T_a, M, t, d, q) = \begin{cases} [d, 0] & \mathrm{Start}(T_a) \leq t < \mathrm{Finish}(T_a) \\ [d, \mathbf{q}_0(M)] & \text{otherwise} \end{cases} \tag{3}$$

Computing $\mathbf{d}(M, t)$ and $\mathbf{q}(M, t)$. Underlying a TÆMS model is a simple state-based computation. Each method has an initial maximum quality $\mathbf{q}_0(M)$ and duration $\mathbf{d}_0(M)$ so we define $\mathbf{q}(M, 0) = \mathbf{q}_0(M)$ and $\mathbf{d}(M, 0) = \mathbf{d}_0(M)$. If there are no non-local effects, then $\mathbf{d}(M, t) = \mathbf{d}(M, t-1)$ and $\mathbf{q}(M, t) = \mathbf{q}(M, t-1)$. If there is only one non-local effect with M as a consequent $\mathsf{nle}(T, M, e, p_1, p_2, \ldots)$, then $[\mathbf{d}(M, t), \mathbf{q}(M, t)] \leftarrow e(T, M, t, \mathbf{d}(M, t-1), \mathbf{q}(M, t-1), p_1, p_2, \ldots)$. If there is more than one non-local effect, then the effects are applied in the order mutex, then enables, then facilitates.

A mutually exclusive resource coordination problem. A Mutex Coordination Problem (MCP) is thus defined as a TÆMS objective task structure, which can be represented as a tuple $\langle \mathbf{A}, \mathbf{T}, \mathbf{M}, \tau, \mathbf{E}, \mathbf{R}, \rho, \delta \rangle$ where

- $\mathbf{A}$ is a set of agents
- $\mathbf{T}$ is a set of TÆMS tasks
- $\mathbf{M}$ is a set of TÆMS actions (methods)
- τ is a mapping from tasks to sets of subtasks and/or actions
- $\mathbf{E}$ is a set of non-local effects between elements of $\mathbf{T}$ and/or $\mathbf{M}$

- **R** is a set of uniquely named resources
- ρ is a mapping from a subset of actions **M** to resources **R**.
- δ is a mapping from every action in **M** to the agents that are capable of executing that action.

For the simple case of mutex relations only, we simplify this to $\langle \mathbf{A}, \mathbf{T}, \mathbf{M}, \tau, \mathbf{E}, \delta \rangle$ by adding mutex relationships to **E** between every pair of methods that accesses the same uniquely named resource. However, we can not do this in general, see Section 8.

For the purposes of evaluation, a *solution* S to an MCP can be represented abstractly as a set of after-the-fact *schedules* for each agent, indicating the Start and Finish times for each action. We can then calculate the finish time of S, Finish(S), as the latest action finish time, and the amount of work done as the sum of the work done in each action. In our experiments, we create a specific model of a hospital with fixed *maximal* values for **A** (the Ancillaries); **T** and **M** (all the possible hospital tasks and their component actions), δ (mapping from actions to the single appropriate ancillary), and **E** (task interrelationships including both enables, facilitates, and *mutex*). We then generate "patients" by choosing a subset of tasks; generate different "historical" models by removing (simplifying) the number of enables and facilitates relationships, and examine the effect of the mutex coordination mechanism by manipulating the probability of mutex relationships (see Section 6).

4. Generalized Partial Global Planning (GPGP)

GPGP is a domain independent *scheduling* coordination approach. The term "planning" in the name is historical, arising from Durfee's PGP work. In the modern AI view of a continuum between planning and scheduling, both GPGP and PGP focus on the scheduling side—on the relative ordering and absolute temporal placement of actions. The GPGP approach makes several architectural assumptions on the agents involved. Most important of these is that the agent represents its current set of intended tasks using the TÆMS task structure representation language (see Sections 3 and 6.1).

An agent using the GPGP approach provides a planner or plan retriever to create task structures that attempt to achieve agent goals, and a scheduler that attempts to maximize utility via choice, serialization, and absolute temporal location of basic actions in the task structure. Each GPGP *mechanism* examines the changing task structure for certain situations, such as the appearance of a particular class of task relationship, and responds by making local and non-local *commitments* to tasks, possibly creating new communication actions to transmit commitments or partial task structure information to other agents. The set of coordination mechanisms is extendible, and any subset or all of which can be used in response to a particular task environment situation. Initially, GPGP defined the following five coordination mechanisms based on Durfee's PGP:

Updating non-local viewpoints. Each agent detects the possible coordination relationships and then communicates the related task structures. A *coordination relationship* is simply a task interrelationship (e.g. enables, facilitates, etc.) that extends between the task networks of two different agents. Detecting the existence of such relationships is domain dependent. In a domain such as distributed sensor networks, possible coordination relationships are detected geographically with respect to physical sensor locations[9]. In an application such as financial information gathering[11] possible relationships are recorded before a partial plan is distributed to multiple agents. In the hospital scheduling problem, the set of "possible relationships" are well-known medical domain knowledge, and are based on the particular set of tests ordered by the examining doctor and recorded by the nursing unit.

Communicate results when they will be used by others. For example, if the results of task A at agent A will enable the execution of task B at agent B, then actually send those results when they become available. In our previous GPGP studies, we modeled the performance of communicating whenever it seemed advantageous versus only when tasks had been committed to, with respect to environmental features such as rate of dynamic change, message size, and likelihood of distraction [8, 10]. The standard result communication mechanism also sends notifications when a result *cannot* be delivered due to some failure, and when an agent believes all of its work on a joint goal has been completed (similar to the Cohen & Levesque model of teamwork [20]).

Handling simple redundancy. When more than one agent wants to execute a redundant method, one agent is randomly chosen to execute it and send the result to the other interested agents. This can lead to more complex load-balancing mechanisms for handling redundancy [7, 5]. Like all the mechanisms, this one can be switched on or off for different domains or parts of a domain—sometimes redundancy is *desirable*.

Handling hard relationships (A must come before B) from the predecessor side. A is the "predecessor" task, B is the "successor". The idea used in PGP and generalized in GPGP is that the agent with the predecessor task will commit to a completion time locally, and then transmit the commitment to the agent responsible for B. Note that this is *not* the only way to handle this relationship (see below).

Handling soft relationships from the predecessor side. A "soft" relationship exists between A and B if when A is executed before B, the execution of B will be perhaps faster or will return better results, but it is not strictly necessary. A simple example is sorting versus searching: sorting facilitates searching, but sorting is not strictly necessary before searching. In this PGP generalization, again the agent with the predecessor task commits to a completion time and transmits the commitment to the successor.

The most important thing about the GPGP approach is that it assures the generality of the mechanisms, because each mechanism is specified as a response to

some pattern in a TÆMS task structure. Although the specific task structure differs from task instance to task instance and domain to domain, these coordination relationship show up over and over again in different locations in each new domain. Thus the GPGP approach allows us to apply the five Durfee mechanisms to domains other than distributed vehicle monitoring (such as randomly generated problems, distributed data processing[24], choosing organizational forms[6], local area network diagnosis [29], or hospital patient scheduling and information gathering as discussed in this paper). The only limitation is the reliance on a TÆMS specification of the underlying task.

However, just because a mechanism can be applied to any domain does not mean that it should be. One might consider if there is an optimal set of mechanisms (usually very hard to determine except in very simplified domains), or consider if some set of mechanisms either models current practice (an organizational modeling perspective) or performs better than the current set of mechanisms. For example, any particular coordination relationship, enables for example, can engender *many* possible, reasonable coordination mechanisms. For example, if a task TA at agent A enables task TB at agent B, one could (this is a highly abridged list):

- Have A commit to a deadline for TA (the existing original PGP-inspired mechanism)
- Have A send the result of TA ("out of the blue", as it were) to B when available
- Have B request that A complete TA by some deadline
- Have B poll for the completion of TA (Our model of current hospital practice)

The point is that there are many possible responses (coordination mechanisms) to a particular coordination relationship. The choice of a response may be conditioned on expected performance, or on organizational structure or social norms that constrain an agent's responses.

5. Applying GPGP to Hospital Scheduling

5.1. The Minimal Required Coordination Mechanisms

To achieve the goal of finishing all the tests, some coordination mechanisms are required. One is the communication of results. To know which tests to do, and to whom to communicate the results, the agent needs to know the corresponding part of the task structure, so the mechanism of updating non-local viewpoints is also needed. Since there are no redundant tasks, we do not need the mechanism for handling redundancy. Handling hard and soft relationships is not strictly required, since (as we just discussed) an agent can just wait around until it gets the needed results, and then begin the successor task (and in fact, this is slightly better than current actual practice in the hospital). However, we will show later that coordinating over these relationships can improve performance. So, the simplest agent in handling the hospital scheduling problem will have just two coordination mechanisms, and will represent our model of current hospital organizational practice.

5.2. A New Coordination Mechanism for Resource Constrained Problems

In the hospital scheduling problem, some (but not all) tests can only be done with the patient physically being there. Thus a resource constraint problem arises, where the patient becomes a crucial non-sharable resource for different agents. A mechanism that can respond to this situation is not included in the five mechanisms initially defined based on PGP, and cannot be created by simply combining some of those mechanisms. Therefore we describe a new resource-constraint coordination mechanism. The new mechanism uses a simple multi-round (but not multi-stage) negotiation process which is not optimal but has good "flow" properties since at least one agent is free to stop meta-level communication and begin domain work at each round [19]. Our experience is that in highly dynamic environments with large amounts of uncertainty, excessive time taken to find an optimal or near-optimal schedule is wasted [5]. Other assumptions that we make include that the agents are cooperating with one another, use the same bid evaluation strategy, and that their utility measures are comparable (here, we are usually dealing with the amount of time saved, which is a comparable measure).

When several agents try to use the same non-sharable resource at overlapping times, only one agent can actually get the resource and execute its work. The others who failed to get the resource waste this time unit and this effort. The idea behind the resource-constraint coordination mechanism is that when an agent intends to execute a resource-constrained task (i.e. the task is scheduled locally), it sends a directed bid of the time interval it needs and the local priority (expressed as the effect on local utility) of its task (we'll describe how this is computed later). After a communication delay, the agent knows all the bids given out by the other agents at the same time as its own bid. Since all the agents who bid have the same information, if they all use the same commonly accepted rule to decide who will get the time interval, they can get the same result on this round of bidding. The agent who won will keep its schedule and execute that task at the pre-determined time interval it bid, and everyone else will mark this same time interval with a *DON'T* commitment and never try to execute a related resource-constrained task in that interval unless the owner gives it up. All the agents who didn't get their time intervals at this round will reschedule and bid again. The detailed process is as follows:

1. (Re)Schedule. Compute the best local schedule given all current information and DO and DON'T commitment constraints (initially, none). No task can start sooner than the communication-delay.

2. For the next resource-constrained task in the schedule, send out bids indicating the time interval desired and the corresponding priority.

3. Get information about the bids received (after some communication-time delay), determine who gets which time interval (i.e., the bid with highest priority, see Section 5.2.1), and make DO or DON'T commitments as appropriate, updating the local scheduling information. Also process retractions of commitments from agents that are giving up some time interval. We'll discuss how to handle

multiple *non-overlapping* bids on the same resource in Section 5.2.2. If your local, resource-constrained task can no longer be scheduled now, go back to Step 1. If your task is a winner, go on to Step 4 at the appropriate time.

4. Monitor the execution of the task in question. When you receive others' bids, you still mark the time intervals with the appropriate DON'T commitments for the scheduler just in case something goes wrong, the world changes, and you need to reschedule. If you find that the task has significantly changed its execution time in the schedule, notify the other agents that you give up the time interval. If the task is finished, stop this process.

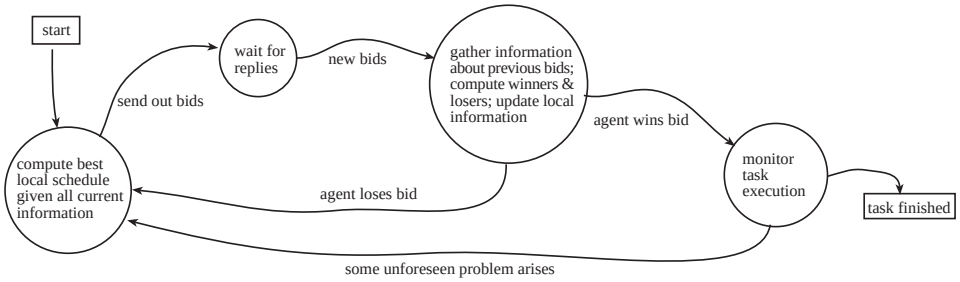

Figure 2. Finite state machine for the bid process

For example, let the communication delay be one time unit, and let there be three agents and tasks as follows: Agent A has task A11 of duration 3 and task A12 duration 3; Agent B has task B11 of duration 2; Agent C has task C11 of duration 4, and task C12 duration 1. A11, B11 and C11 have a shared resource-constrained relationship.

time 0: each agent communicates only the task structures related to the shared resource constraint (the GPGP "updating non-local viewpoints mechanism")

time 1: each agent makes its own local schedule.

schedule: Agent A: A11(2–4), A12(5–7); agent B: B11(2–3); agent C: C12(2–2). Assume that task C11 is not in the local schedule of C for some external cause, such as C11 not being enabled or Agent C otherwise currently believes it does not have to do C11. The decisions about each schedule are made locally.

Agent A sends out a bid for time interval 2–4 with priority 3. (The agent decides the priority of a task according to its local view, thus it might be "incorrect" w.r.t a non-existent global view).

Agent B sends out bid for time interval 2–3 with priority 4.

time 2: each agent gathers information about the bidding at time 1.

Agent A finds that the other agent won time interval 2–3, so it marks this time interval occupied and then tries to reschedule. The new schedule is A12(4–6), A11(7–9). Agent A sends out a bid for 4–6 with priority 3.

Agent B finds it won, so it keeps its schedule and starts its execution of B11 at time slice 2.

Agent C, reconsidering, puts C11 in schedule, C11(3–6) and executes C12. C sends out bid for 3–6 with priority 2.

time 3: Agent A finds that it won, so it keeps its schedule and will execute A12 starting in time slice 4.

Agent C finds that it lost, so it marks time interval 2–6 occupied and reschedules. The new schedule is C11(7–10). C sends out a bid for time interval 7–10.

time 4: Agent C won, it keeps its schedule. C will wait until time slice 7 to begin its execution.

To put this mechanism to practical use, we need to decide two more things. One is how an agent computes the priority about its local task. The other is how to decide if an agent wins or loses according to the priorities of all the agents.

5.2.1. Determining the bid winner When using this mechanism to handle resource constrained problems, we will assume that each agent will be honest, i.e.. they will not cheat on the priority calculation. Since different agents may have different goals with respect to each other and to any global goals, the performance of such a system can be worse than that of a centralized system with fixed centralized goals. In many domains such as hospital scheduling (or telescope observation scheduling [1]), however, we *cannot* centralize scheduling because it would take away the authority of each unit over the day-to-day control of its own activities. Our distributed approach matches with the existing human organizational structure. It also allows each unit to attempt to optimize slightly different measures, as may be used by administrators to evaluate human unit-level performance. In our work, we try to find the best priority function for agents that can minimize the finish time of the whole task. We found that (a) the number of coordinated relationships with the task (e.g. enables or facilitates) and (b) the start time of that task should both be considered when computing the priority of a task. The reason for considering the coordinated relationships is apparent: a task that enables or facilitates many other tasks should have higher priority. Each bid includes not only the requested time interval, but also a priority calculated locally by the bidder that captures this task relationship information. Since in the hospital patient scheduling task, all of the tasks need to be done, we concentrate solely on the effect on task durations. If T is the task we will bid on, and $\mathcal{E}$ is the set of tasks that T enables, and $\mathcal{F}$ is the set of tasks that T facilitates, then

$$\text{Priority}(T) = \frac{\sum_{E \in \mathcal{E}} \mathbf{d}(E) + \sum_{F \in \mathcal{F}} \phi_d \mathbf{d}(F)}{\mathbf{d}(T)}$$

where $\mathbf{d}(T)$ is the estimated duration of a task T, and ϕ_d is the "power" of the facilitates relationship (for example, a 50% reduction in duration would be $\phi_d = 0.5$).

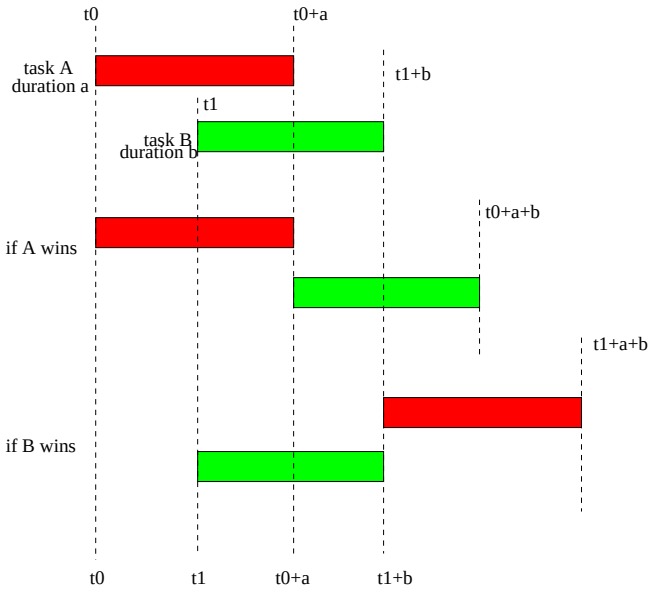

Figure 3. Why an earlier task has higher priority

The following example (Figure 3) will show why a task starting earlier should have higher priority.

Assume task A and B both need a mutually exclusive resource and plan to start at $t0$ and $t1$ respectively, and $t0 < t1$. If A wins, both tasks can finish at time $t0 + a + b$, if B wins, the finish time is $t1 + a + b$. The reason is that the time interval between $t0$ and $t1$ is wasted. So, all other things being equal, an early starting task should have higher priority.

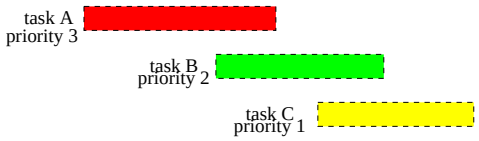

Figure 4. Example: Three bids for a single resource. Should we choose tasks A and C, or just task A?

To summarize, then, the winner of a bidding round is the earliest task from the set with the highest priority. Although it happens rarely in practice, if there is still a tie (exactly the same priority and start-time) we break it in a consistent manner (for example, a time-indexed ring of the agent's names, or any similar traditional distributed tie-breaking scheme).

5.2.2. Dealing with multiple non-overlapping bids It's easy to know who gets the time interval if there are only two tasks competing for the resource. How about when there are more than two tasks? For example see Figure 4. Since task A has highest priority (all other things being equal), it will win. But how about task C? Since task B didn't win its time interval (because of the conflict with A), there will be no conflict if C is also executed at its time interval. Thus, should task C win too? The results of those two different methods are shown in Figure 5.

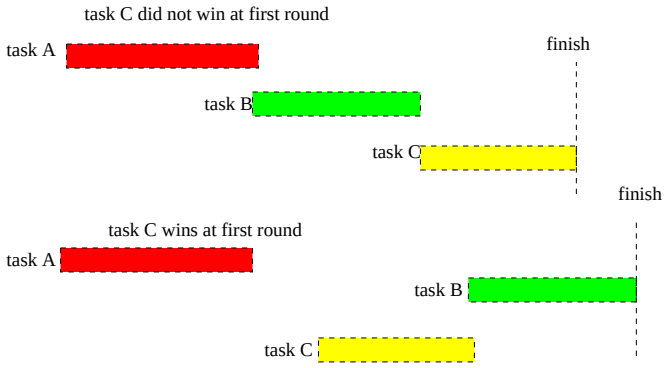

Figure 5. The result of different methods to deal with multiple, non-overlapping bids

There will be two problems that occur when we let task C also win its time interval. First, task B actually has higher priority than C, but it lost the competition of A vs B vs C. In this situation, if a task loses in one round, it may have to start its execution very late even though it has higher priority than the intervening tasks. Second, the time after task A finishes and before task C starts is wasted: nobody uses this time interval.

The advantage of allowing more than one task to get its time interval is that it can reduce the number of rounds of negotiation. When the cost of communication or rescheduling is high compared with the cost of execution, this method will get better performance. In our experiments, we used the first method where only one task wins at each round.

In domains other than this one, the priority calculation must take into account other characteristics over which agents may express preferences (usually represented as a utility function). One of the features of TÆMS is that the indication of these utility-affecting changes is done in a precise, quantitative manner. However, since a peculiarity of this problem is that all tests must be done, complex reasoning about cost and benefit tradeoffs is not discussed here (but see [5]).

6. Evaluation

Since final quality is not so important in the hospital scheduling problem (tasks are either done or not done—this is a "goal-oriented domain" as opposed to a "worth-oriented" domain [26]), we use the average task finish time and the actual work

done by all agents to measure the performance of each agent. The task finish time shows us how quickly a patient can finish all the tests, i.e. the latency of the system. The actual work done by all agents can be used to determine the throughput of the whole system.

6.1. TÆMS *Task Structure Modeling*

In both goal-oriented and worth-oriented domains, the important components of an environment's structure include the *attributes* of world states over which an agent may express preferences, and the effects of agent actions on those attributes. As shown earlier in Section 3, The TÆMS language framework has been developed to model goal- or worth-oriented environments in the support of agent development. TÆMS has been used in two ways: as part of an agent's internal computational representation of its environment, and as a way to carefully evaluate agent architectures and algorithms. As an evaluation tool, we use three levels of models (generative, objective, and subjective—only the subjective models are "known" by individual agents) to simulate and carefully control agent percepts and the effects of their actions. This level of control allows us to perform (for example) paired response studies, sensitivity analyses, and average-case execution time studies. The value of TÆMS is in providing a language with which to model complex, worth-oriented environments including the *quantitative* specification of continually changing real-valued attributes. Through techniques such as reusable sets of non-local effects and attribute accumulation functions TÆMS also allows *qualitative* reasoning to take place, such as the triggering of particular GPGP coordination mechanisms.

6.2. Generative Hospital Patient Scheduling Model

Our generative model of this environment focuses on the number and distribution of patient tests. The episode generator, a simulation based on the description of Ow's hospital, has a fixed number of nursing units and ancillaries, and a fixed set of test templates that correspond to taking x-rays, physical therapy, blood tests, etc. These templates contain statistical duration distributions on how long it takes to complete each action. Then, we postulate for each patient a distribution of tests (instantiated from the set of templates). Thus when each patient arrives, linked to that patient is a unique task structure of what needs to be done to that patient. In other words, to simulate the process of hospital scheduling, first we create a big task tree template which represents all the possible ancillaries, tests and interrelationships between these tests (figure 6). Note that some, *but not all* of the tasks require the physical presence of the patient.

For each simulated patient, we stochastically choose ancillaries and tests from this big task template tree, to generate a single MCP instance. We use three independent variables to control the probability of:

- non-local relationships (relationships between ancillaries). This is a measure of how interdependent medical tests are.

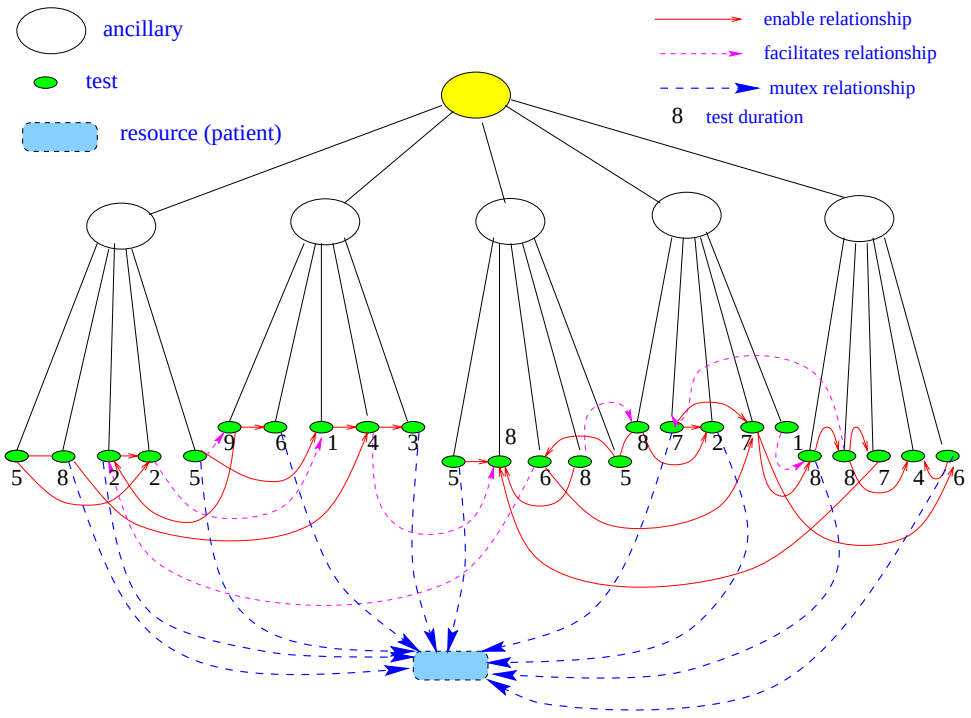

Figure 6. Task tree used in experiments

- local relationships (relationships within a single ancillary). This is a measure of how complex an ancillary's individual scheduling problem is.

- patient resource relationships (how many tests require the patient to be physically present). For example, taking an x-ray requires the physical presence of the patient, but the other tasks involved in an x-ray procedure (e.g., developing and analyzing the film) do not require the patient's presence.

Those variables have values between 0 and 1. A possibility equal to 1 means it will definitely contain all the relationships from the big task tree template (but remember that this is *not* "all possible relationships").

Subjectively, we allow the portions of the task structures associated with each ancillary to arrive simultaneously at each ancillary. Since there is only one possible ancillary for each test (no redundancy), we do not simulate the trivial process of sending each task from the nursing unit to the unique associated ancillary. From Ow's studies and our discussions with hospital personnel, there is no opportunity for negotiation at task assignment (e.g. multiple ancillaries vying for the identical task). These opportunities only appear as the patient treatment plan is serialized and actions are fixed in time.

6.3. Experimental Results

In general, we conduct paired-response experiments on two groups of agents:

Simple Agents follow our model of current hospital practice as it pertains to patient scheduling. They use the "communicate results" and "non-local views" coordination mechanisms only, and handle inter-ancillary task coordination relationships by simply waiting for the enabling task to be completed.

Cooperating Agents begin with the Simple Agents model, and add to that coordination mechanisms for handling hard and soft ordering constraints (e.g. enables and facilitates) by a predecessor commitment (from the original PGP gang of five mechanisms) and the new mutex coordination mechanism for negotiating the mutually exclusive patient resource.

All of the simulated medical patient treatment plans are done with an average of 3 ancillaries and an average of 3 medical tests in each ancillary. With each set of given possibilities, we run 40 repetitions. We compute the average difference in finish time of those tests. Given the hypotheses that the cooperating agents use *more* time than the simple agents, we use the non-parametric Wilcoxon matched-pairs signed-ranks test to calculate the probability with which we can reject that hypotheses, i.e. show that the coordinating agents actually use *less* time. We also compute the average difference in actual work done by the two groups of agents.

From those results, we can show:

- The average finish time of cooperating agents is always less than that of simple agents.

- With an increase in non-local relationships, the finish time performance of co-operating agents first decreases then increases with respect to that of simple agents (see the left side of Figure 7). The reason is because an agent has to use more effort to deal with the non-local relationships while it cannot always get enough benefit from this cooperation. Our set of mechanisms is significantly better than just the two simplest mechanisms at the $\alpha = 0.05$ level for structures with 80% or more of the full number of inter-ancillary relationships.

- Performance in finish time of cooperating agents increases with the increase of patient resource constraints (see the left side of Figure 10. Our mechanisms are again significantly better at 80% of the full number of relationships, when the non-local relationships are normal.

- Throughput performance of cooperating agents increases with the increase of patient resource relationships. See the right side of Figures 8 through 10.

- The change in non-local relationships has no apparent affect on throughput performance; our mechanisms have a consistent advantage at all levels. See Figure 7.

Each of the following figures has two graphs: the amount of time saved on the left, and the amount of work saved on the right. Higher numbers are better from the standpoint of our set of coordination mechanisms. The vertical y-axis indicates the measured average time or work saved using our approach (higher is better), and the x-axis indicates the independent variable (probability of inter-ancillary relationships or probability of patient resource relationships).

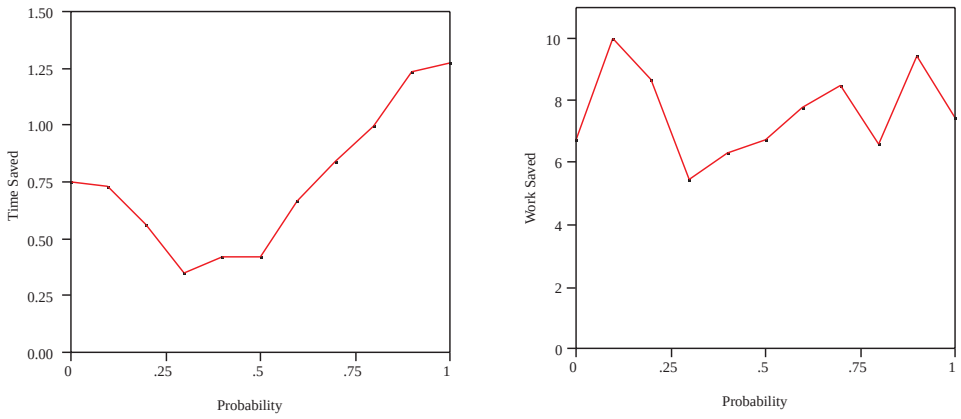

Figure 7. [y-axis] less time (left) and work (right): [x-axis] increasing the probability of inter-ancillary relationships (with all patient resource relationships present)

Experiments run with randomly generated task structures, as expected, support these observations. A random task structure is generated using randomized HTN

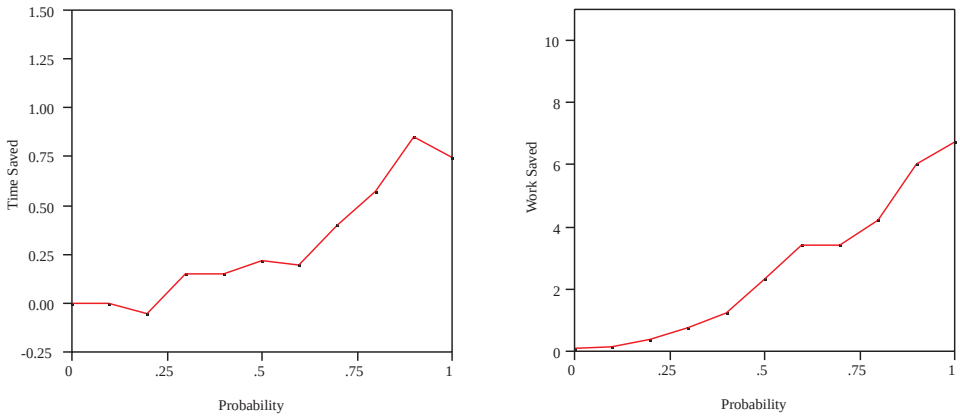

Figure 8. [y-axis] less time (left) and work (right): [x-axis] increasing the number of patient resource relationships (using no inter-ancillary relationships)

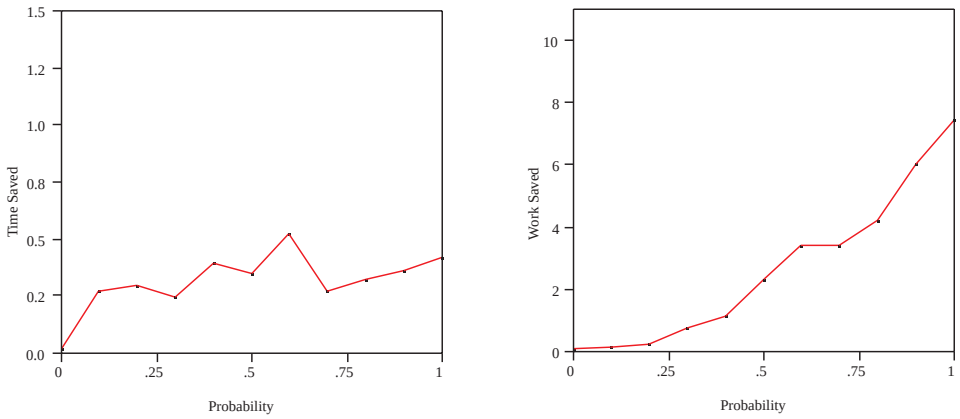

Figure 9. [y-axis] less time (left) and work (right): [x-axis] increasing the number of patient resource relationships (holding constant a 50% chance of standard inter-ancillary relationships)

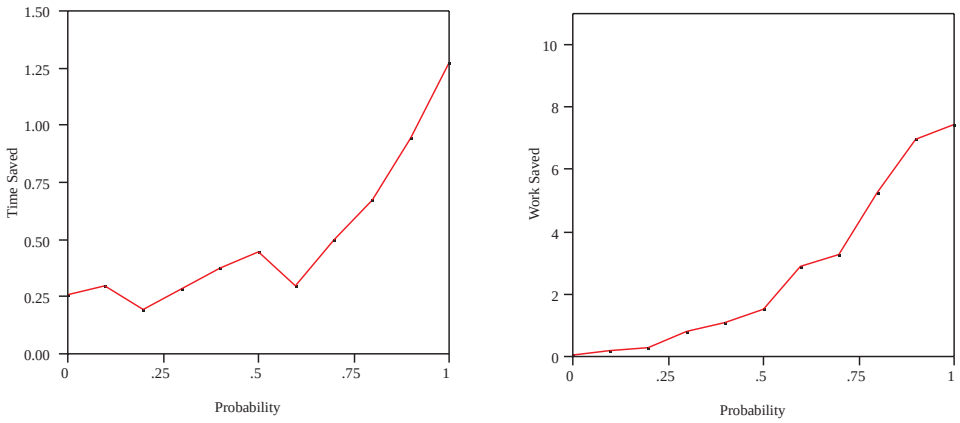

Figure 10. [y-axis] less time (left) and work (right): [x-axis] increasing the number of patient resource relationships (while all standard inter-ancillary relationships are always present)

building with average tree depth, breadth, possibility of interrelationships and random assignment of tree leaf actions to agents (see [5] for detailed information). We are currently evaluating performance of the mutex coordination mechanism in an internet information gathering domain and extending it to handle a more generalized notion of a resource constraint (see Section 8).

7. Related Work

Liu & Sycara[21] developed a coordination mechanism called Constraint Partition and Coordinated Reaction (CP&CR). In this mechanism, each resource is assigned to a resource agent which is responsible for enforcing capacity constraints on the resource, and each job is assigned to a job agent which is responsible for enforcing temporal precedence (i.e. enables in TÆMS) and release data constraints (like the early-start-time commitments in TÆMS) with the job. The job agents are similar to the agents in this paper, responsible for finishing their own jobs. They later extend the CP&CR to a new coordination mechanism Anchor&Ascend to deal with the Constraint Optimization Problem [22]. This mechanism also assigns resources to a resource agent. It identifies a bottleneck resource, and identifies the agent that is responsible for this resource as the Anchor agent. The Anchor agent makes the best local subsolution first, then all other agents make their own subsolutions while trying not to violate the subsolution of Anchor agent. The difference between our mechanisms and those two mechanisms is that CP&CR and Anchor&Ascend are partly centralized with respect to each resource. When dealing with hospital scheduling problem, the disadvantage of that style of solution is that it lets an agent outside of an ancillary to directly affect the ancillary's local schedule.

Ow, Prietula, and Hsu also develop a organization structure for hospital scheduling problem [23]. Their organization contains unit agents which are responsible for

monitoring the test schedule of each patient as well as collecting and disseminating test requests to the ancillary systems, and the ancillary agent which is responsible for collecting the test requests sent to it and schedules them. We have used this same organizational structure. Ow's paper doesn't give any details about how each ancillary agent deals with the resource constraint problem, only that each ancillary agent will broadcast the times that have been blocked out to all other ancillaries (somewhat similar to the first step of the mechanism we propose here). Ow's mechanism also does not explicitly handle interactions between tests at different ancillaries. The set of GPGP coordination mechanisms described here easily respond to these inter-ancillary interactions as well as eliminate the full broadcast communication.

A great deal of interesting work has also been done on multi-agent meeting scheduling [27, 14]. This work has focussed on coordinating the resources and typically has not dealt with multiple hard and soft task interrelationships.

Simpler approaches to resource coordination can be found in traditional distributed systems. For example, a centralized coordinator approach (one agent decides which time slots should go to which agent) could be used with a centralized coordinator chosen via the bully algorithm [30]. All such centralized solutions violate the organizational constraints of distributing these decisions at the hospital ancillaries.

The existing mutex coordination mechanism is somewhat related in its flow properties to traditional algorithms such as CSMA (Carrier Sense Multiple Access) used in networking [31]. In this algorithm the shared resource is the physical local network (i.e. Ethernet). In our example, if an agent loses a bid for a time slice, the agent reschedules the task, finds the next time slice, and bids again (similar to nonpersistant CSMA).

8. Extensions to Limited Capacity Resources

In this paper we only considered the simplest type of resource constraint, a mutually exclusive resource. However, TÆMS can model a wider range of resources [29, 5]. In particular, we can model limited capacity/bandwidth resources as a pair of resource/task relationships: uses and limits. A task uses some capacity of a resource and an overused resource imposes some limits on the offending tasks.

For example, take the situation of a low-bandwidth communication link as described in [29]. Two agents have a diagnosis method that uses the low-bandwidth link. When more than one of these diagnosis methods are executed at the same time, the link is saturated and the durations of the methods are lengthened. We represent this situation as shown in Figure 11; the two new NLE's are defined as follows:

$$\mathsf{uses}(T_a, R, t, d, q, \alpha) = \begin{cases} [d, q+\alpha] & \text{Start}(T_a) < t < \text{Finish}(T_a) \\ [d, q] & \text{otherwise} \end{cases} \tag{4}$$

$$\mathsf{limits}(R, M, t, d, q, \sigma, \phi_d) = \begin{cases} [\phi_d d, q] & Q(R,t) > \sigma \\ [d, q] & \text{otherwise} \end{cases} \tag{5}$$

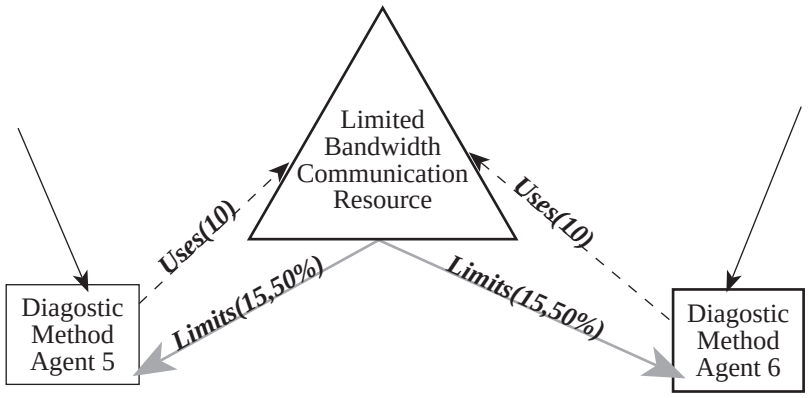

Figure 11. Example of two methods sharing a limited resource of capacity 15. In this example, if both methods execute in temporally overlapping time periods, the durations of each method will be lengthened by 50%.

The NLE uses indicates the amount of limited resource capacity used by the method with the parameter α. In Figure 11 each diagnostic method uses 10 units of the communication resource's capacity (the arc labeled uses(10)). The NLE limits indicates that the resource has maximum capacity (saturation) σ and that beyond that point duration is affected by percentage ϕ_d (alternately, ϕ_d could be defined as a function of the level of oversaturation). In Figure 11 the communication resource has a saturation point $\sigma = 15$, so the resource will not be saturated if either diagnostic method uses the resource alone, and will be saturated if the two methods happen to overlap in their execution. The effect will only be active during the overlap, and will cause the duration of both methods to be increased (in this example) by 50%. Note that when methods can be interrupted, the lengthening of method durations due to a blocked resource is not associated with lengthening the amount of continuous computation—the blocked method can be interrupted and other computations performed until the method is no longer blocked.

We are generalizing the mutex coordination mechanism to handle such limited resources. Two major differences have been discovered. First, compared to mutex resources, limited capacity resources require more sophisticated reasoning about multiple requests, since often *several* requests can be handled in a given time slice. Secondly, with many different resources, hard and soft task relationships may be harder to handle since they may link otherwise independent resources used by different parts of a single task. These differences have been addressed by communicating more resource state information, ordering resource-constraint coordination resolution before hard and soft task interrelationship resolution, and changing the multi-conflict algorithm as shown in Figure 5.

This type of resource is useful in modeling domains such as internet information gathering, where certain external information servers and databases can be modeled as such limited resources. We can represent an abstract information gathering problem as follows. Each agent is responsible for several *queries* $\mathcal{Q}$. Each query

$Q \in \mathcal{Q}$ comprises a set of *clauses* $\mathcal{C}$. These clauses are related to one another via enables or facilitates relationships, as can be determined by a query planner (see for example, Knoblock's work [17]). Multiple query plans for one query are possible. For example, if there are multiple sets of data that must be combined, then different orderings of *join* operations will take different amounts of time. Furthermore, each clause may be retrieved from one of several different *sources*, each with its own access costs, reliability, quality, etc. The sources are, or course, limited resources, joined to each clause by a pair of uses and limits relationships (represented by a single double arrow in Figure 12).

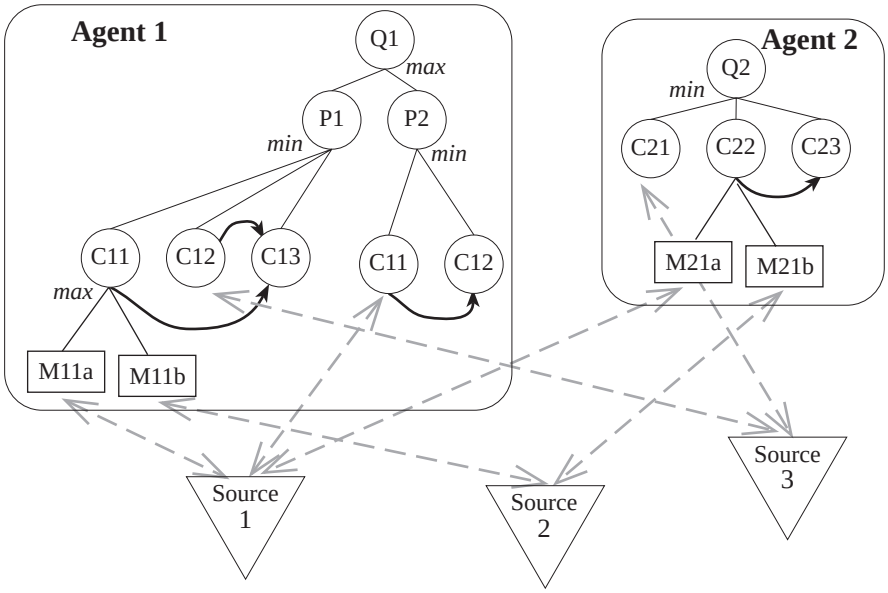

Figure 12. Abstract TÆMS representation of a simple information-gathering problem: 2 queries, one with multiple plans, and several clauses, some of which could be fulfilled at possibly multiple, limited-capacity sources.

This work is not being carried out in simulation, but rather in a real-time distributed agent toolkit called DECAF [15]. In order to reduce broadcast communication and to make the system scalable, we introduce a *resource manager agent*, which enforces access to a mutex or limited-capacity resource. Initially, the resource manager agent centralizes the bid processing (for some resource), carrying out the decision process as described in this paper and modified in the previous paragraph. However, we are also looking into *multi-stage* negotiation[4], where the resource manager agent bounces a conflict *back* to the agents involved for further negotiation. This will also allow the development of solutions for non-cooperative systems.

9. Conclusions and Future Work

In this paper we presented a model of the Hospital Scheduling Problem in TÆMS, and applied the GPGP approach to solving this problem. Hospital scheduling is interesting not only because it is a real problem, but because it is an inherently distributed problem, not amenable to a centralized solution because of the human organizational authorities involved. An extremely similar problem is that of scheduling telescopic observations at university observatories in the U.S. (each university wishes final control over the schedule of their own telescope, although they totally cooperate with other universities in scheduling observations)[1]. Some of the unique features of this particular problem that are very different from our previous work is that all the tasks need to be done, and the state of being "done" is either yes or no, rather than a set of continuous characteristics. There are no deadlines.

We did not represent travel times (and neither do the real hospitals), however, there is an implication of this downplay of travel times—the hospital will have to change its coordination and scheduling mechanisms if it has to face significant travel delays with *more than one ancillary*. This could happen if, for example, it is a smaller hospital and cannot afford its own MR scanner or CAT scanner and doctors take to prescribing MR and CAT scans at another hospital. Patients requiring *both* MR and CAT scans now have a significant facilitates effect in their task structures to do both scans nearly consecutively (potentially affecting the coordination structures of *both* hospitals). Other than changing the *representation*, we could handle this situation without changing the existing set of coordination mechanisms. At this point, our hospital model results are only from simulated data, drawn from Ow's study, and not from actual hospitals. We are currently in the process of applying this resource negotiation approach to real, non-simulated systems, such as satellite antennae scheduling, in collaboration with a commercial company. We are also interested in increasing the hospital model's realism by using more detailed data from real hospitals.

This paper also demonstrated the use of the GPGP approach in solving this problem. In this environment, the mechanisms to communicate task structures and partial results are necessary, while the mechanisms of handling hard and soft relationships are optional. By providing a solution without changing any existing mechanisms, we demonstrated again the generality of the GPGP approach. By limiting ourselves to just the two necessary mechanisms, we were able to model current practice fairly closely.

We then focussed on the specification and implementation of a new GPGP coordination mechanism oriented toward handling resource constraint relationships between tasks at different agents. The approach was that of a simple single stage, multi round cooperative negotiation, which has good flow properties and low overhead, which is important in an environment where the task mix may change dynamically with new patients, and where there is uncertainty over exactly how long procedures will take. We are aware of the utility of more complex multi-stage negotiations (e.g. [4]), and intend to examine the utility of such higher-overhead, higher-payoff approaches. The new mechanism we have developed can now be ap-

plied to any environment with resource constraints. Finally, we demonstrated the effect of the new coordination mechanism on performance in different environments. The experimental results show that the new mechanism increases the performance of agents by both decreasing patient stays and increasing throughput when there are many inter-ancillary relationships.

Acknowledgments

The authors would like to acknowledge the help of Terry Harvey and the reviewers in helping to make this paper more readable. This effort was sponsored by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory Air Force Materiel Command, USAF, under agreement number F30602-97-1-0249. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. This material is also based on work supported by the National Science Foundation under Grant No. IIS-9733004.

Notes

1. The full set of quality accumulation functions, including alternate definitions for AND and OR, is discussed in [5].

References

1. J. Bresina, W. Edgington, K. Swanson, and M. Drummond. Operational closed-loop observation scheduling and execution. In *Proceedings of the AAAI-96 workshop on Theories of Planning, Action, and Control*, 1996.
2. C. Castelfranchi. Commitments:from individual intentions to groups and organizations. In Michael Prietula, editor, *AI and theories of groups & organizations: Conceptual and Empirical Research*. AAAI Workshop, 1993. Working Notes.
3. Philip R. Cohen and Hector J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42(3):213–261, 1990.
4. S. E. Conry, K. Kuwabara, V. R. Lesser, and R. A. Meyer. Multistage negotiation for distributed constraint satisfaction. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6), November 1991.
5. Keith S. Decker. *Environment Centered Analysis and Design of Coordination Mechanisms*. PhD thesis, University of Massachusetts, 1995. `http://dis.cs.umass.edu/~decker/thesis.html`.
6. Keith S. Decker. Task environment centered simulation. In M. Prietula, K. Carley, and L. Gasser, editors, *Simulating Organizations: Computational Models of Institutions and Groups*, pages 105–131. AAAI Press/MIT Press, 1997.
7. Keith S. Decker and Victor R. Lesser. An approach to analyzing the need for meta-level communication. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 360–366, Chambéry, France, August 1993.
8. Keith S. Decker and Victor R. Lesser. A one-shot dynamic coordination algorithm for distributed sensor networks. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 210–216, Washington, July 1993.
9. Keith S. Decker and Victor R. Lesser. Quantitative modeling of complex computational task environments. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 217–224, Washington, July 1993.

10. Keith S. Decker and Victor R. Lesser. Designing a family of coordination algorithms. In *Proceedings of the First International Conference on Multi-Agent Systems*, pages 73–80, San Francisco, June 1995. AAAI Press. Longer version available as UMass CS-TR 94–14.
11. Keith S. Decker and Katia Sycara. Intelligent adaptive information agents. *Journal of Intelligent Information Systems*, 9(3):239–260, 1997.
12. Edmund H. Durfee and Victor R. Lesser. Using partial global plans to coordinate distributed problem solvers. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, August 1987.
13. K. Erol, D. Nau, and J. Hendler. Semantics for hierarchical task-network planning. Technical report CS-TR-3239, UMIACS-TR-94-31, Computer Science Dept., University of Maryland, 1994.
14. L. Garrido and K. Sycara. Multi-agent meeting scheduling: Preliminary experimental results. In *Proceedings of the 2nd International Conference on Multi-Agent Systems*, Kyoto, June 1996. AAAI Press.
15. J. Graham and K.S. Decker. Towards a distributed, environment-centered agent framework. In *Proceedings of the 1999 Intl. Workshop on Agent Theories, Architectures, and Languages (ATAL-99)*, 1999.
16. N. R. Jennings. Commitments and conventions: The foundation of coordination in multi-agent systems. *The Knowledge Engineering Review*, 8(3):223–250, 1993.
17. C.A. Knoblock. Building a planner for information gathering: A report from the trenches. In *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems (AIPS96)*. AAAI Press, 1996.
18. A. Kumar and P.S. Ow. A study of distributed problem solving for patient scheduling. In *Proc. ORSA/TIMS*, Washington, D.C., 1988.
19. Brigitte Lâassri, Hassan Lâassri, and Victor R. Lesser. Negotiation and its role in cooperative distributed problem solving. In *Proceedings of the Tenth International Workshop on Distributed AI*, Texas, October 1990. Also COINS TR–90–39.
20. Hector J. Levesque, Philip R. Cohen, and José H. T. Nunes. On acting together. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 94–99, July 1990.
21. J. Liu and K. Sycara. Distributed problem solving through coordination in a society of agents. In *Proceedings of the Thirteenth International Workshop on Distributed AI*, pages 190–206, Seattle, WA, July 1994. AAAI Press Technical Report WS-94-02.
22. J. Liu and K. Sycara. Exploiting problem structure for distributed constraint optimization. In *Proceedings of the First International Conference on Multi-Agent Systems*, pages 246–253, San Francisco, June 1995. AAAI Press.
23. P. S. Ow, M. J. Prietula, and W. Hsu. Configuring knowledge-based systems to organizational structures: Issues and examples in multiple agent support. In L. F. Pau, J. Motiwalla, Y. H. Pao, and H. H. Teh, editors, *Expert Systems in Economics, Banking, and Management*, pages 309–318. North-Holland, Amsterdam, 1989.
24. M.V. Nagendra Prasad, K. S. Decker, A. Garvey, and V.R. Lesser. Exploring organizational designs with TÆMS: A case study of distributed data processing. In *Proceedings of the Second International Conference on Multi-agent Systems*, Kyoto, Japan, December 1996.
25. M.V. Nagendra Prasad and V.R. Lesser. Learning situation-specific coordination in generalized partial global planning. In *AAAI Spring Symposium on Adaptation, Co-evolution and Learning in Multiagent Systems*, Stanford, March 1996.
26. J. S. Rosenschein and G. Zlotkin. *Rules of Encounter: Designing Conventions for Automated Negotiation among Computers*. MIT Press, Cambridge, Mass., 1994.
27. S. Sen and E. Durfee. Unsupervised surrogate agents and search bias change in flexible distributed scheduling. In *Proceedings of the First International Conference on Multi-Agent Systems*, pages 336–343, San Francisco, June 1995. AAAI Press.
28. R. Simmons. Structured control for autonomous robots. *IEEE Trans. on Robotics and Automation*, 10(1), February 1994.
29. Toshiharu Sugawara and Victor R. Lesser. On-line learning of coordination plans. Computer Science Technical Report 93–27, University of Massachusetts, 1993.
30. A. Tanenbaum. *Modern Operating Systems*, chapter Synchronization in distributed systems. Prentice Hall, 1992.

31. A. Tanenbaum. *Computer Networks*, chapter The medium access sublayer. Prentice Hall, 1994.
32. T. Wagner, A. Garvey, and V. Lesser. Complex goal criteria and its application in design-to-criteria scheduling. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, Providence, July 1997.
33. M.P. Wellman and J. Doyle. Modular utility representation for decision-theoretic planning. In *Proc. fo the First Intl. Conf. on Artificial Intelligence Planning Systems*, pages 236–242, June 1992.
34. M. Williamson, K. S. Decker, and K. Sycara. Executing decision-theoretic plans in multi-agent environments. In *AAAI Fall Symposium on Plan Execution*, November 1996. AAAI Report FS-96-01.