Lateral and Hierarchical Partial Centralization for Distributed Coordination and Scheduling of Complex Hierarchical Task Networks*

Mark Sims, Hala Mostafa
Bryan Horling, Haizheng Zhang
Victor Lesser and **Daniel Corkill**
University of Massachusetts
Multi-Agent Systems Laboratory
{msims,hmostafa,bhorling,hzhang,lesser, corkill}@cs.umass.edu

John Phelps
Honeywell Laboratories
3660 Technology Drive
Minneapolis, MN 55418
john.phelps@honeywell.com

Abstract

We describe a new approach to coordinating the scheduling and execution of a complex hierarchical task structure distributed among a set of agents. Our approach decomposes allocation, scheduling, and monitoring into manageable local pieces that are coordinated with one another. Each agent is assigned mediator responsibilities for multiple tasks in the task structure so that each task has a mediator. We also establish mediator responsibilities to deal with task interrelationships that span portions of the task network located in different agents. In this mediator organization, each mediator is responsible for a small, tractable portion of the global scheduling and execution process. Mediators coordinate their allocation and scheduling decisions with other mediators, both hierarchically and laterally. Mediators communicate relevant local-agent activities using an abstract partial-order representation of resource availability and flexibility and combine these abstractions to make informed scheduling decisions for their portion of the global task structure. We detail how our approach operates in challenging hierarchical task settings, such as those being scheduled and executed in the DARPA COORDINATORS program, and we discuss how our approach differs from related techniques.

Introduction

We present a solution to the problem of scheduling and executing a hierarchical task structure that is distributed among a set of agents, none of which have a global view. Our solution must meet the following requirements:

- be responsive to unforeseen dynamics
- reschedule quickly
- have low interaction among agents
- localize the responses to dynamics as much as possible.

*This material is based upon work supported by the DARPA/IPTO COORDINATORs program and the Air Force Research Laboratory under Contract No. FA8750–05–C–0030. The views and conclusions contained in this document are those of the authors, and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

We use a heuristic approach that employs a negotiation protocol with the following key aspects:

- partial centralization
- lateral and horizontal mediation
- abstraction
- partial order schedules.

Our approach partially centralizes allocation, monitoring, and scheduling by assigning mediation responsibilities for tasks to the agents in the system. This divides the task network into a static structure of local task mediator responsibilities organized both hierarchically and laterally. This structure decomposes the global scheduling problem so that each task's mediator is responsible for a small, tractable portion.

An important aspect of our approach is the unique combination of lateral and hierarchical mediators serving as multi-agent schedulers that use abstractions in their interactions. There are two types of mediators: Task Managers and Chain Managers. Both are assigned to the agents in the system and each agent can have multiple mediator responsibilities. A Task Manager is assigned to each task in the task hierarchy and is responsible for collecting information about that task's subtasks, handling commitment requests, and scheduling operations for the task. A Chain Manager handles enablement interrelationships that exist laterally across portions of the task network. An enablement relationship exists between two tasks if the first must be accomplished in order for the second to be executed. It is possible that a task that enables another may itself be enabled by some other task. Thus, these interrelationships can form chains of enablements that span large numbers of tasks and agents. For each chain a single Chain Manager is responsible for obtaining a complete view of the chain and performing scheduling operations involving it. When the operations of a task and chain manager intersect, the Task Manager is subservient to the Chain Manager since the Chain Manager has the greater context for understanding the non-local implications of scheduling tasks in the chain.

We use abstractions to limit the scheduling process for each task at every level in the hierarchy and reduce the complexity of the multi-agent scheduling problem. The abstraction that mediators construct is based on a partial-order representation of agent activities. It summarizes the possible

ways that tasks may be achieved. By collecting partial-order schedule alternatives from subordinates, a mediator is able to build a composite representation of the activities of each agent that contributes to solving the mediator's task. The reduction in complexity from using abstractions comes at a cost, however. The abstraction process merges the activities of an agent into a single continuous activity with flexibility to move within a larger time window. The result is that the agent may appear able to perform longer duration tasks than it actually can and unable to perform short duration tasks that might fit between existing tasks. It is possible that agents can recover from the former problem since the failure to form a commitment for a long duration task will be caught and a new agent will be requested to perform the task or a new, shorter task will be requested. The latter problem, however, cannot be recovered from since the abstraction process obscures feasible schedules, making it impossible for a requesting agent to know that it can request the short duration task. Future work will address this problem.

Overview of Our Approach

To mitigate agents' incomplete views, our approach employs a multi-step scheduling process. The first phase identifies and assigns mediators based on the structure of the task network and occurs prior to any commitment formation. Once mediators are assigned, a commitment acquisition process begins with a top-down, tentative commitment phase in which mediators send out "rough ideas" of when subtasks should be performed. A rough idea for a subtask gives the largest possible time window during which that task could be performed. In response, lower level mediators respond with alternatives for when they are able to perform their tasks given the rough ideas. The result of this process is that each mediator in the hierarchy obtains a space of possible solutions for its task.

The final phase establishes firm commitments based on alternatives obtained in the tentative commitment phase. Because firm commitments are formed asynchronously, a mediator may need to perform additional local search over the alternatives, making it likely that a mediator can resolve conflicts without involving higher-level mediators. If this is not possible, backtracking to a higher mediator will occur.

Along with firm commitments, mediators send values that reflect the commitments' importances from a perspective with more context. These values are a mechanism for locally resolving conflicting requests for an agent's resources. As a result, a mediator may decommit from a commitment which then triggers additional search and backtracking. A similar recovery process is followed if an agent discovers that it cannot satisfy a commitment during execution.

Finally, execution is based on partial-order schedules which allow agents to avoid unnecessary rescheduling due to minor changes if sufficient flexibility in the schedule exists.

The Approach at Work

Our hierarchical task networks are represented in C_TAEMS (Boddy *et al.* 2005), a refined subset of the TAEMS (Decker & Lesser 1993) representation language. Initially, each agent receives a portion of the task structure as its local view that includes interrelationships from nonlocal tasks that affect the agent's local tasks. Figure 1 shows the global view of a simple, four-agent problem. Figure 2 shows the subjective view of Agent 1.

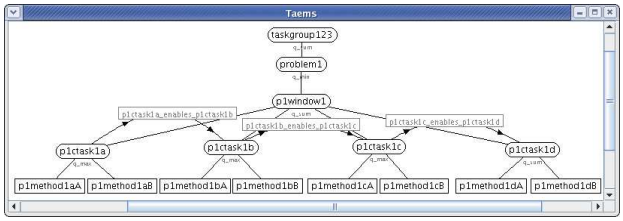

Figure 1: Example global task structure.

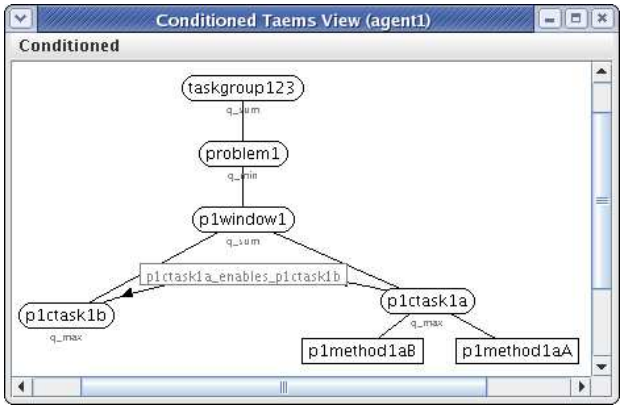

Figure 2: Agent1's initial subjective view

A task may appear locally in many agents' subjective views. For instance, task *p1window1* is visible to all agents. The agents know that each sees the task because the labels of the agents that have a task in their task structures are stored in a *visible-to* list with the task. However, no agent sees all of the task's subtasks. Long enablement chains of nonlocal interactions, involving many tasks and agents, also characterize the environment. Due to agents' limited views, no agent has a complete view of an entire chain.

Definitions

Problem-Level A problem-level task is a subtask of the root task in a hierarchical task network.

Method-Level A method-level task (or simply a method) is a leaf in the hierarchical task network.

Intermediate-Level An intermediate-level task is any task in the network that is neither the root, a problem-level task, nor a method.

QAF As seen in the figures throughout this paper, under each non-leaf task in the task network is a quality accumulation function (QAF). The QAF specifies how the quality of a task's subtasks combine to determine the quality of the task itself (Horling *et al.* 1999). The important QAFs are

MIN (the quality of a task is the minimum quality of any of its subtasks), **MAX** (the quality of a task is the maximum of the quality of its subtasks), **SUM** (the quality is the sum of the subtasks' qualities), and **EXACTLY_ONE** (exactly one subtask must be performed and the quality of the task is equal to that subtask's quality). Note that any subtask that is not performed has zero quality.

Partial Order Schedule A partial order schedule consists of a set elements with associated constraints derived from a TAEMS task structure. Each element contains a method. Each method is associated with a single agent and multiple methods may pertain to the same agent. Further, a method in an element can be an actual task from that agent's local task structure or an abstraction of that agent's activities used in a more abstract task structure. The constraints on a single element include its *release* and *deadline* specifying its earliest start time and latest finish time and possibly a *start_at* constraint specifying the particular time the method must start. *Precedence* constraints between methods are derived from enables and facilitates interrelationships between those methods in the TAEMS structure.

A partial order schedule, as opposed to a linear schedule, allows subsystems to exploit the flexibility of an agent's TAEMS structure. The partial order schedule avoids linearization that is not required by the precedence relations assuming the elements are specific, low-level methods. If each element of a partial order schedule is an abstraction of multiple activities, there is additional flexibility from the delayed decision of how those activities should be elaborated. The coordination component uses the constraints to schedule activities and negotiate time windows for actions. The execution component uses the constraints to determine if rescheduling is needed if a method takes longer than expected, or if the downstream actions can simply be shifted within the partial order schedule.

Rough Idea/Tentative Commitment A rough idea is the largest possible time window during which a subtask can be performed from the perspective of the parent task's manager or the subtask's Chain Manager. In practice a mediator requesting tentative commitments forms a rough idea from the fixed constraints in its task structure and the constraints from any tentative commitments it has received from other mediators. For example, a Task Manager will use the earliest start time and deadline on its task as the bounds of the window in the tentative commitment for each of its subtasks.

The manner in which rough ideas are formed implies that multiple rough ideas sent out by a mediator for tasks will possibly violate constraints between those tasks. This is not a problem. The rough ideas are used merely to gather information from other mediators as to when their tasks can be achieved. The requesting mediator will use the returned information to find schedules that do not violate constraints.

Alternative An alternative is a partial order schedule with associated quality and cost information passed from one mediator to another to summarize the sender's possible contribution to a larger schedule. A mediator sends multiple alternatives to another in response to a rough idea/tentative commitment request. The elements of alternatives returned by method-level task managers consist of agents' actual method-level tasks. The elements of alternatives returned by higher-level managers consist of abstractions of agents' activities. Therefore, an alternative provides not only quality, cost, and duration information about how a task can be achieved, but also abstract information about the contributions of other agents.

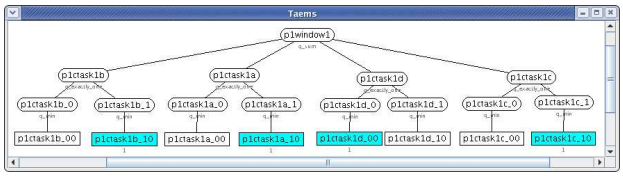

Figure 3: Example of the abstract structure used to schedule the subtasks of a task.

To understand how mediators generate and propagate alternatives, it is instructive to see how Task Managers schedule in response to tentative commitments. At the method-level a Task Manager uses the Design-to-Criteria (DTC) scheduler (Wagner, Garvey, & Lesser 1998) to perform a local scheduling process that takes the agent's existing commitments and schedule into account.

DTC outputs an ordered list of linear schedules. For each that contains the requested method, the Task Manager generates a partial order schedule containing the required method. The set of these schedules represents the different ways in which the agent can achieve that method.

When an intermediate or a problem-level manager receives alternatives from subtask managers, it performs a different scheduling process. The manager first generates a four-level task structure that captures the abstract information contained in the alternatives. Figure 3 shows the abstract view of agent 1 in its capacity as Task Manager of *p1window1*. The root is the managed task (*p1window1* in this example). The second level consists of the managed task's subtasks. The third level tasks correspond to alternatives for the second level tasks. In other words, for each alternative received for a subtask, the manager creates a new subtask of the subtask. Because only one alternative can be used to satisfy a subtask, second-level tasks have **EXACTLY_ONE** QAFs.

Fourth-level methods are derived from the partial order schedules associated with third-level tasks by merging all elements in a schedule associated with the same agent into a single method. Each such method is a composite of the earliest start time (EST) and deadline (DL) information in the elements from which it is derived. The new EST of a composite method is $min(EST_1, EST_2, ...EST_n)$ where EST_i corresponds to the earliest start time of the i^{th} element before merging. The deadline of the composite is $max(DL_1, DL_2, ...DL_n)$. The new duration is simply the sum of the durations of the original elements.

As mentioned in the introduction, merging all elements associated with the same agent into a single method obscures the fact that there may be slack time between elements and

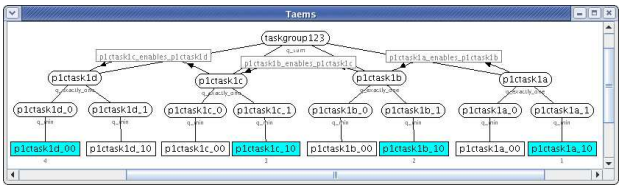

Figure 4: A Chain Manager's abstract task scheduling structure.

that the agent may not be able to perform tasks whose durations are less than the combined slack time. In future work, we will explore situations in which a finer-grained abstraction may be necessary (as discussed in Status and Future Work).

Although each third-level task in Figure 3 has one fourth-level method, a third-level task generally has multiple subtasks.

DTC cannot schedule multiple agents' tasks. It treats all tasks as if they belonged to one agent which obscures the potential for parallelism. To enable us to schedule multi-agent task structures, we created the Multi-Agent DTC (MADTC) construct. MADTC uses a task structure transform that enables DTC to treat agents as TAEMS consumable resources (Horling *et al.* 1999) and schedule overlapping methods.

While Task Managers schedule to generate alternatives for a single task, Chain Managers use MADTC on four-level, multi-agent abstract views to generate alternatives for entire chains. Figure 4 shows agent 3's Chain Manager view. The root has a SUM QAF to allow the Chain Manager to schedule the whole chain or only part of it. The second level tasks are the chain members. The third and fourth levels are as for Task Managers.

Refined Commitment A refined commitment for a subtask is a firm commitment that is sent out by a mediator after it has collected and processed alternatives from other mediators. The time window of a refined commitment is contained within the rough idea in the earlier tentative commitment for that task. Included with the refined commitment is its value determined from a more complete context (see below).

Commitment Value In many real-world situations, agents cannot necessarily accomplish all commitments they are asked to perform. As such, it is important during the scheduling process for agents to understand the relative values of commitments, so that they can make appropriate decisions when tradeoffs are unavoidable.

Because no agent necessarily has a complete view of how tasks and methods interact, it is not possible to compare two requests without additional information. For example, suppose an agent must choose between performing some method A that is expected to produce quality 10 and B with quality 1. Although A is superficially better than B, it is only by evaluating those options in context that their true value may be known. If B is at the root of a critical chain, or if A is below a **MAX** QAF that has alternatives with greater expected quality, then B may in fact be the globally appropriate choice. In both of these situations, however, the information needed to make the decision is not necessarily available in an agent's local task structure.

This lack of knowledge means that the agent must rely on the requesting manager to correctly value the commitment that it is proposing. A Chain Manager will know if B is a critical member of the chain, and a Task Manager will know if A has higher-quality alternatives. So, if those two managers know the value of the commitments they have received, they can use that information along with the structural knowledge they have accumulated to determine the value of the commitments they request. Commitment valuation is therefore a recursive process, with individuals relying on information present in the commitments they have themselves received. The ultimate source of this information comes from the highest-level manager, which can determine the global expected quality for its task based on the expected quality information present in the subtask alternatives it chooses to pursue.

The manner in which a Task or Chain Manager assigns the value of the commitments it proposes depends on 1) the value of the commitment it is responding to, 2) the relative qualities of the alternatives it has received and 3) the requested task's QAF (Task Manager) or position in the chain (Chain Manager). For example, suppose a Task Manager with a **SUM** QAF has been requested to satisfy a commitment with value v. The value for the set of subtask commitments C is computed as:

foreach c **in** C

$$c.value = min(v, c.expected\_quality)$$
$$v = max(v - c.value, 0)$$

A similar function distributes value evenly among subtasks in the case of a **MIN**, and another places the majority of value on the alternative with greatest expected quality in the case of a **MAX**. A Chain Manager will assign the same value to all tasks in the chain that are required to perform the task that was originally requested. Different valuations will produce different behaviors. For example, by spreading value to more than one subtask under a **MAX** the manager can obtain a certain level of redundancy.

Because all valuations are recursively derived from the expected quality that occurs at the root task, they will be appropriately normalized such that the agent can correctly rank the various requests that it receives. This allows it to produce local schedules that indirectly take into account global objectives, so that team-level needs are respected despite the fact that the individual agent may not have the detailed structural knowledge required to infer those needs itself.

Initialization

During the initialization phase Task Managers are chosen for each task in the hierarchical task network. They then expand their local views to include information about subtasks not originally visible to them. Chain Managers are also selected for each enablement chain and expand their views to include entire chains.

Task Manager selection is based on the C_TAEMS task's *visible-to* list. Agents in the list know its contents, obviating

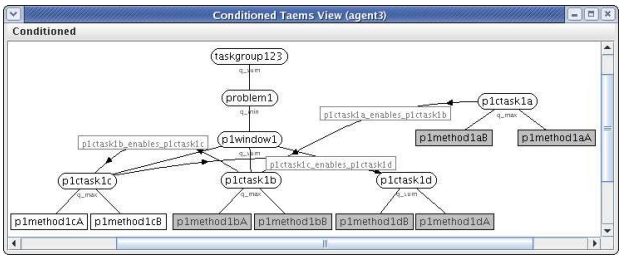

Figure 5: Agent3's subjective view after subtask retrieval and chain discovery

the need for a negotiation or an election.[1] The task's name is a seed to a pseudorandom process that deterministically selects an agent from the list to be that tasks's manager.

After an agent recognizes that it is the Task Manager of a task, it sends a request to each agent in the *visible-to* list for an abstraction of any locally-managed subtasks of the manager's task. This subtask retrieval process provides the Task Manager with a complete list of its subtasks.

Returning to Figure 2, the pseudorandom process determines that agent 1 is the Task Manager of *p1window1*, and agent 1 begins subtask retrieval to obtain information about all four tasks under *p1window1*.

Chain Manager selection is similar to Task Manager selection. An agent may be a Chain Manager if its local view contains a task with incoming but no outgoing interrelationships. Such a task is a chain sink. Because multiple agents may see this task, a pseudo-random process similar to that for Task Managers deterministically chooses the Chain Manager from the sink's *visible-to* list.

When an agent realizes that it is a Chain Manager, in the simplest scenario it discovers the chain members by following the enablement links between nodes, acquires chain member information in a process similar to that used by Task Managers to obtain subtask information, generates an ID for its chain, and informs chain members of the chain ID and the manager's name. It is possible, however, that multiple chains interact. This occurs, for example, if a task enables more than one task or a task has multiple incoming enables relationships. In this case, there will be multiple chain sinks each with its own Chain Manager. These Managers compete for managing the nodes upstream of the intersection. Each Chain Manager sends a management request to the Task Manager of the task at the intersection. This Task Manager grants the first request it receives and declines the rest. The winning Chain Manager explores the rest of the chain starting from the task(s) enabling the task at the intersection until it either reaches a node without incoming enablements or competes in another race. The losing Chain Manager stops exploring its chain and includes the task at the intersection as the first chain member even though it is managed by a different Chain Manager. This information will be used later during the negotiation over tasks in the interacting chains.

Figure 5 shows that agent 3 as Chain Manager has a complete view of the chain in the global structure after chain discovery.

[1] In general, however, this will not be the case since the *visible-to* list is specific to the C_TAEMS representation.

Negotiation Protocol

Once subtask and chain information is in place at the appropriate Task and Chain Managers, task scheduling and negotiation begins. The pseudocode in Algorithms 1 through 8 illustrates the major pieces of the protocol. In the pseudocode TM refers to Task Manager, CM to Chain Manager. The Task Manager of a problem-level task initiates the negotiation protocol with a vertical process. This manager generates rough ideas for each of its subtasks and sends out tentative commitments to each subtask manager. Ignoring the presence of enablement chains, at each level a manager receiving a rough idea repeats the process using the constraints in the tentative commitment as the bounds of its rough ideas. The process bottoms out at the manager of a method. This manager schedules its task locally in a process that takes existing commitments into account to form alternatives and passes them to the sender of its tentative commitment. Intermediate Task Managers collect alternatives from below and schedule using them to generate alternatives to pass up. When the problem-level manager receives alternatives from below, it schedules to generate and send down refined commitments for a subset of its subtasks. These propagate down the tree and eventually reach method-level nodes, resulting in the methods being scheduled for execution.

Algorithm 1 Task Manager Start Up

```
if problem-level then
    generate rough ideas
    send tentative commitments to subtask managers
    wait for alternatives to be returned
else
    wait for tentative commitment message
end if
```

Algorithm 2 TM Handle Tentative Commitment(msg)

```
if method-level then
    schedule to generate alternatives
    pass alternatives to msg sender
else
    if (chain member ∧ msg not from CM) then
        Forward msg to CM
    else if (not chain member) ∨ (chain member ∧ msg
    from CM) then
        generate rough ideas
        send tentative commitments to subtask managers
        wait for alternatives to be returned
    end if
end if
```

When a tentative commitment arrives at a task that is part of an enablement chain, a horizontal process interrupts the vertical one. The manager receiving the tentative commitment forwards it to its Chain Manager. In response the Chain

Algorithm 3 TM Handle Alternatives(msg)

```
if haven't received alts from all subtask managers then
   queue msg
else
   Schedule to merge queued alternatives
   if chain member then
      send merged alts to Chain Manager
      wait for alternative from Chain Manager
   else if problem-level then
      Extract refined commitments from highest rated schedule
      send refined commitments to subtask managers
   else
      send merged alts to higher manager
   end if
end if
```

Algorithm 4 TM Receive Refined Commitment(msg)

```
if chain member ∧ msg not from CM then
   forward msg to CM.
else if intermediate Task then
   send refined commitments to subtask managers
else
   schedule method for execution
end if
```

Algorithm 5 TM Handle Alt from Chain Manager(msg)

```
send alternative to upper task manager
```

Algorithm 6 CM handle forwarded Tent. Com(msg)

```
generate rough ideas
send tentative commitments to all chain members
```

Algorithm 7 CM Handle Alternatives(msg)

```
if haven't received alts from all chain members then
   queue msg
else
   Schedule to merge queued alternatives
   find highest rated schedule
   send single chain-consistent alternative to each chain member.
end if
```

Algorithm 8 CM Handle Forwarded Refined Com(msg)

```
if haven't received refined coms from all chain members
then
   queue msg
else
   Extract refined commitments from highest rated schedule
   send refined commitments to all chain members
end if
```

Manager sends tentative commitments to all chain members in order to collect alternative information from them and make chain-consistent scheduling decisions. If a chain member is actually managed by a different Chain Manager due to interacting chains, that chain member forwards the tentative commitment request to its actual Chain Manager which is then responsible for scheduling that node and its downstream tasks (not shown in the pseudocode). After the Chain Manager has received alternatives from all chain members, it schedules the chain to find chain-consistent alternatives for each chain member to pass up to the manager that sent the original tentative commitment. In our current implementation, the Chain Manager sends a single alternative to each chain member. It does this by finding the most highly rated chain schedule and sending each chain member an alternative that corresponds to its part in the schedule. This ensures that when higher-level task managers request refined commitments of chain members, all constraints within the chain will be respected. We are exploring options for sending more alternatives to each chain member without allowing constraints to be violated.

When an upper-level Task Manager receives a single alternative from a chain member, it treats the alternative exactly as it would any other. When it eventually sends a refined commitment to the chain member, the chain member forwards the commitment to its Chain Manager. After the Chain Manager has received forwarded refined commitments from all members, it sends out refined commitments to all Chain Members. The reason for this extra step is to ensure that the chain can indeed be executed.

We now illustrate the steps of the protocol using part of a trace from our example scenario. In the following, a Task Manager of a task is referred to as *TM-task-name*. Initially, *TM-problem1* sends a tentative commitment to *TM-p1window1* with a rough idea spanning the interval [20 120] which represents the earliest start-time and deadline of *p1window1*. *TM-p1window1* then sends tentative commitments with the same rough idea to *TM-p1ctask1a* through *TM-p1ctask1d*. The steps below pertain to *TM-p1ctask1a*, but are similar for its three sibling Task Managers:

1. *TM-p1ctask1a* at Agent 1 receives from *TM-p1window1* also at Agent 1 the tentative commitment request with rough idea [20 120].
2. The vertical process is interrupted. *TM-p1ctask1a* realizes that *p1ctask1a* is chain member and forwards the rough idea to its Chain Manager (agent 3).
3. Agent 3 as Chain Manager, having received similar messages from other chain members, uses the rough ideas as constraints, generates new rough ideas, and sends tentative commitments to chain members.
4. *TM-p1ctask1a* receives the new tentative commitment from the Chain Manager and uses it to constrain its scheduling. The vertical process resumes and *TM-p1ctask1a* sends tentative commitments to *TM-p1method1aA* and *TM-p1method1aB* both with [20 120] as the rough idea.
5. *TM-p1method1aA* and *TM-p1method1aB* schedule locally and send alternatives to *TM-p1ctask1a*. In general,

the scheduling process would take their existing commitments into account by not sending alternatives that would violate those commitments. In this example, there are no preexisting commitments. The alternative from *TM-p1method1aA* consists of a single element with an expected quality of 40 and an expected duration of 20. The alternative from *TM-p1method1aB* also consists of a single element with an expected quality of 32 and an expected duration of 16.

6. *TM-p1ctask1a* reschedules its subtasks using these alternatives, resulting in alternatives of when *p1ctask1a* can be done. It sends the three possible alternatives to the Chain Manager: one that includes both methods and has expected quality 40 (because the QAF under *p1ctask1a* is a **MAX**), one that contains just *p1method1aA* with quality 40, and one that contains just *p1method1aB* with quality 32.
7. After the Chain Manager receives alternatives from all members, it reschedules, resulting in alternative schedules for the whole chain. From the highest quality schedule, the Chain Manager extracts each member's chain-consistent alternative and sends them out. The alternative that the Chain Manager sends to *TM-p1ctask1a* indicates that *p1ctask1a* can be performed any time between time 20 and time 116.
8. *TM-p1ctask1a* receives its alternative from the Chain Manager and forwards it to *TM-p1window1*.
9. *TM-p1window1* reschedules and sends alternatives to *TM-problem1*. In response, *TM-problem1* sends a refined commitment for *p1window1* with an earliest start time (EST) of 20 and a deadline of 120. The reason the refined commitment has the same bounds as the tentative commitment *TM-problem1* originally sent is that *p1window1* is the sole subtask under *problem1* in this example. The refined commitment's value is 44 (*p1ctask1a* supplies most of the quality. The remainder comes from its siblings).
10. In response to the refined commitment *TM-p1window1* sends a refined commitment to *TM-p1ctask1a* also with an EST of 20 and deadline of 120. The reason it does not use the [20 116] range is that *TM-p1window1* is unaware of the constraints due to the chain. From its perspective [20 120] is just as good as [20 116]. The value on the commitment is 40 to match *p1ctask1a*'s expected contribution.
11. *TM-p1ctask1a* forwards the refined commitment from *p1window1* to the Chain Manager as do its sibling Task Managers. In response the Chain Manager sends finalized commitments with their values. The commitment for *p1ctask1a* has an EST of 22, a deadline of 63, and a value of 44. The time constraints differ from those in the forwarded commitment because the Chain Manager has a complete view of the chain and must ensure that all tasks in the chain can be accomplished. The value of the commitment is different because the Chain Manager understands that if *p1ctask1a* is not performed, the tasks it enables also cannot be performed. Therefore, its value from the Chain Manager's perspective is higher than that from *TM-p1window1*'s perspective.
12. After *TM-p1ctask1a* receives the Chain Manager's refined commitment, it sends a refined commitment to *TM-p1method1aA* only. When the method-level manager receives this commitment, it includes the method in its schedule.

Handling the Unexpected

As with any realistic environment, there will be cases where the agent's expectations will not be met. This happens when local actions do not complete as expected, when subordinates fail to satisfy their commitments, or when the task structure itself changes in response to external or environmental demand. In each of these cases the system must react appropriately to succeed in the new conditions.

Failed Actions

A method deviates from its expected outcome when it takes longer than expected to complete, or when it fails to accrue sufficient quality. This may have both direct and indirect implications; a method taking longer than expected may violate its own local deadline and also indirectly affect other scheduled activities by depriving them of time to execute.

The agent must consequently reason about the larger ramifications of a deviation. The first step in that process is to exploit the flexibility in the partial order representation, which can avoid a potentially expensive rescheduling process. For example, if the remaining actions all have release times later than the current time, no changes may be needed.

We are concerned with agents that can perform only one action at a time. Because of this, although the constraints in the partial order representation may indicate that many actions can start at a particular time, only one may be selected in practice. The process of finding a totally-ordered sequence of actions from the partial order representation that respects all constraints is called *linearization* and is performed by the agent's execution component. Once an action's expected duration has been exceeded, the feasibility of the current local schedule is determined by searching for a valid linearization among the remaining activities.

When creating a linearization, it is not sufficient simply to pick an arbitrary action whose constraints are currently satisfied, because the order of actions in a linearization, together with their durations and deadlines, affects how many of them can be included in it.

Because of interactions such as this, the search for a valid linearization must explore the space of sequences characterized by the partial order representation. If the existing linearization is not valid, the search progresses in a recursive manner using branch and bound techniques. A search path is bound if at any point there is insufficient time (relative to the latest applicable deadline) to perform the remaining activities, or if any constraints on the most recently placed action have been violated. Once a linearization is found, it is stored along with the representation for future use.

If a linearization cannot be found, or if a completed action does not accrue sufficient quality, the current schedule is invalidated, and a more involved repair process starts. The remaining actions are passed to the local scheduler, which

produces a new schedule consisting of a subset of those actions. Unlike the linearization search, this process may remove actions that cannot be performed, either because the agent no longer has the resources to do so or because higher valued commitments used the needed resources. Commitments that are no longer satisfied are passed to the appropriate local Task Manager which reports that failure to the Task Manager that requested the commitment.

The requesting Task Manger tries to resolve the problem locally. It searches the alternatives it obtained during the scheduling and commitment acquisition process to determine if any of the remaining alternatives can satisfy the constraints imposed by the commitment in question. If so, it sends out new finalized commitments. Otherwise, the manager informs its higher-level manager that the commitment has failed. That manager then repeats the same process.

The procedure for handing commitments that are rejected is identical to that described for failure.

Related Work

Our work extends Generalized Partial Global Planning (GPGP) (Lesser *et al.* 2004) by using mediator-based rather than pair-wise coordination. Also, in our approach agents reason about and obtain commitments for joint goals. Our approach also shares partial centralization aspects with Optimal Asynchronous Partial Overlay (OptAPO) (Mailler & Lesser 2005). The major difference is that in OptAPO, mediators are dynamically created and interact at the same level of detail. In our approach, mediators are statically defined based on the task structure and interact at different levels of abstraction. Durfee and Montgomery (Durfee & Montgomery 1990) also use task hierarchies and abstraction but to solve a different problem of how to avoid negative interactions and capitialize on positive ones among agents. A further difference is that interactions are resolved pair-wise, making it necessary to check that a resolution does not introduce new conflicts. Our partial centralization is designed to avoid this problem. Hunsberger and Grosz solve a similar problem to ours in a centralized manner using a combinatorial auction (Hunsberger & Grosz 2000). Zhang and Lesser also address negotiation chains in a decentralized manner by introducing a pre-negotiation exchange of meta-level information (Zhang & Lesser 2005). Our approach using mediators with expanded contexts is more centralized.

Status and Future Work

The evaluation of our approach to date has focused on relatively simple task structures. Many questions can only be answered by extensive experimentation with large-scale task networks. These include understanding the speed-up gained by decomposing the scheduling problem, how small mediators' abstract views tend to be, how many are solved, and how often mediators must reschedule in response to failure or other changes.

Future work will focus on the abstraction process. Our current method can obscure feasible schedules and make infeasible ones seem feasible. We are considering mechanisms for dynamically changing the granularity of the abstraction. This would provide a way to continue the search process if the first mechanism did not reveal a feasible schedule.

We will also explore other ways of assigning Task Managers. Rather than assign a manager for every task, we can aggregate more tasks under a single manager. This will have the combined effect of reducing the total number of managers that perform scheduling operations and reduce the time for alternatives to percolate up the hierarchy and commitment requests to move down it, potentially at the cost of complicating each mediator's scheduling problem. Key to this will be understanding an appropriate balance of size versus cost.

We are considering mechanisms to recognize more constraints among tasks during initialization to provide tighter rough ideas. Also, our approach to commitment valuation is an approximate technique for handling preemption. We intend to compare it to a more detailed approach involving scheduling both with and without the requested tasks.

References

Boddy, M.; Horling, B.; Phelps, J.; Goldman, R.; Vincent, R.; Long, C.; and Kohout, B. 2005. Ctaems language specification. Unpublished.

Decker, K., and Lesser, V. R. 1993. Quantitative Modeling of Complex Environments. *International Journal of Intelligent Systems in Accounting, Finance and Management. Special Issue on Mathematical and Computational Models and Characteristics of Agent Behaviour.* 2:215–234.

Durfee, E. H., and Montgomery, T. A. 1990. A hierarchical protocol for coordinating multiagent behaviors. In Dietterich, T., and Swartout, W., eds., *Proceedings of the 8th National Conference on Artificial Intelligence (AAAI-90)*, 86–93. Boston, MA, USA: AAAI Press.

Horling, B.; Lesser, V.; Vincent, R.; Wagner, T.; Raja, A.; Zhang, S.; Decker, K.; and Garvey, A. 1999. The TAEMS White Paper. `http://mas.cs.umass.edu/paper/182`.

Hunsberger, L., and Grosz, B. J. 2000. A combinatorial auction for collaborative planning. In *Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS-2000)*, 151–158.

Lesser, V.; Decker, K.; Wagner, T.; Carver, N.; Garvey, A.; Horling, B.; Neiman, D.; Podorozhny, R.; NagendraPrasad, M.; Raja, A.; Vincent, R.; Xuan, P.; and Zhang, X. 2004. Evolution of the GPGP/TAEMS Domain-Independent Coordination Framework. *Autonomous Agents and Multi-Agent Systems* 9(1):87–143.

Mailler, R., and Lesser, V. 2005. Asynchronous Partial Overlay. *Journal of Artificial Intelligence Research*. To appear.

Wagner, T. A.; Garvey, A. J.; and Lesser, V. R. 1998. Criteria Directed Task Scheduling. *Journal for Approximate Reasoning (Special Issue on Scheduling)* 19:91–118.

Zhang, X., and Lesser, V. 2005. Solving negotiation chains in semi cooperative multi-agent systems. Technical Report UMASSD-CIS-TR-2005007, University of Massachusetts Dartmouth.