Coordinating Human and Computer Agents

Keith S. Decker

Department of Computer and Information Sciences
University of Delaware, Newark, DE 19716
decker@cis.udel.edu
[This paper originally appeared in Wolfram Conen, Gustaf Neumann, editors, *Coordination Technology for Collaborative Applications - Organizations, Processes, and Agents*. LNCS #1364, pp. 77–98, Springer-Verlag, 1998.]

Abstract. In many application areas individuals are responsible for an agenda of tasks and face choices about the best way to locally handle each task, in what order to do tasks, and when to do them. Such decisions are often hard to make because of *coordination problems*: individual tasks are related to the tasks of others in complex ways, and there are many sources of uncertainty (no one has a complete view of the task structure at arbitrary levels of detail, the situation may be changing dynamically, and no one is entirely sure of the outcomes of all of their actions). The focus of this paper is the development of support tools for distributed, cooperative work by groups (collaborative teams) of human and computational agents. We will discuss the design of a set of distributed autonomous computer programs ("agents") that assist people in coordinating their activities by helping them to manage their agendas. We describe several ongoing implementations of these ideas including 1) simulated agents and tasks, 2) a real multi-agent system for financial portfolio management, and 3) a real mixed human and computational agent system for concurrent engineering design.

1 Introduction

One operational vision of concurrent engineering is one of a *network* of workstations that enable *concurrent* and *distributed* requirements assessment, design generation, and analysis. It envisions that *groups* of humans and computational 'agents' will *cooperatively* create, resource, evaluate, execute, and monitor design processes or manufacturing plans. Work to date has not directly addressed the problems and *opportunities* that arise from a truly distributed and cooperative environment. The focus of this paper is the development of support tools for distributed, cooperative work by groups of human and computational agents. Patterns of *interrelated task structures* that are partially shared by multiple agents with mixed initiative tasking appear in many military and commercial environments (air campaign planning, concurrent engineering, hospital scheduling, manufacturing scheduling, coordinating large software development efforts). These interrelationships provide not only potential problems to avoid (such as forgetting a required task) but also opportunities to take advantage of (such as doing a task needed by more than one other agent first, or doing a favor for another agent).

The remainder of this paper is organized as follows. First, the overall vision of where we are headed, including an extended example. Next, Section 2 discusses our formal approach to representing coordination problems, called TÆMS, and our approach to interagent scheduling coordination, called GPGP. We have also developed an agent problem-solving environment simulator that allows us to analyze scheduling and coordination algorithms for average performance charactersitcs, the ability to deal with different levels of uncertainty, etc. Section 3 decribes one application of these ideas to a multi-agent system for multi-agent portfolio management. Finally, Section 4 describes an application, just beginning, of these ideas to a mixed human and computational agent system for concurrent engineering design of composite helicopter body panels.

1.1 Where we are heading: an example

In general, each person and computational agent involved in such distributed cooperative work has a set of tasks before them that currently need to be done (identifying targets in several areas, collecting intelligence on each of several disparate targets, etc.). The tasks may come from a local initiative, may be assigned by a superior, or may just be subtasks needed to accomplish current goals. The tasks may be initially perceived as unrelated. We propose to assist users in task selection by developing a User Coordination Assistant Agent (UCAA) that keeps track of a workstation user's current agenda of tasks and presents a possible schedule (ordering) of these tasks according to user- and domain-directed preferences. The idea is not to force a user into the schedule, but rather to provide the user with up-to-date, relevant information so that the user can make an informed decision. Such an agenda is not developed in isolation, but rather through a distributed coordination process using multiple coordination mechanisms triggered by the *coordination relationships* between the task structures of the different agents involved. We also are developing an Agent Coordination Module (ACM) that can be linked-in to provide agenda management services for the coordination of computational agents in accord with the human agents' task-order preferences (see Figure 1).

Such a set of tools will allow the development of systems that can truly support cooperative work involving people at multiple workstations and computational agent assistants by helping to manage and organize the workloads of these agents. Certain tasks will be dependent on other tasks *and* be time critical, and such information for differentiating tasks could be lost in critical situations if only ad hoc coordinating mechanisms are used. It is useful to communicate more information than simply task assignments and task coordination relationships. For example, users and computational agents can request possible due dates for potential tasks before assignment (allowing both better time estimates and load balancing) and request *commitments* to complete certain time-critical tasks (in effect, supporting inter-agent negotiation protocols). Our approach of tying coordination mechanisms to the particular kinds of coordination relationships that exist in a particular task environment provides for a reusable tool foundation (the UCAA and its interactive human interface, the ACM and its interface to a computational agent) and a customizable set of coordination mechanisms for a particular application environment (since no single coordination mechanism would be useful across different environments with different task coordination relationships).

We have already mentioned that one vision of concurrent engineering is of groups of humans and computational 'agents' that will cooperatively create, resource, evaluate, execute, and monitor design processes or manufacturing plans. The technologies that we are developing are aimed squarely at supporting this vision by providing mechanisms to effectively and efficiently *coordinate* such a distributed system. As envisioned, the entire system allows both concurrent activities (carried out by humans and computer programs), and mixed-initiative tasks. Either of these implies there is no fixed, static control of all the human and computational agents in the system. All agents, both human and computational, will be expected to perform multiple tasks—certainly more than one task at a time—in a potentially high pressure, time-sensitive environment. To make things harder, the tasks will not be independent but will be related in potentially complex ways—not only by hard relationships like "A must precede B" but also by softer relationships such as "A would facilitate B", where completing task A will help B by making it easier (quicker) to do, or by making a higher quality result possible, or both.[1] Relationships include not only computational relationships between tasks, but other resources as well. Exclusive access is only one possible resource relationship. Another example is the appearance of diminishing returns in the use of compute servers for large engineering simulations or software design programs, leading to the need for automated load balancing mechanisms.

Figure 1 shows a high-level view of the system described here, with two human user workstations and one autonomous, computational agent (which could be running on either a user's workstation or on a third hardware processor/server).

1.2 Air Campaign Planning Example

We will describe a system in the Air Campaign Planning Domain that consists of several planning tools used concurrently by a JFACC (Joint Forces Air Component Commander), his staff, and associated humans, computational agents, and smart databases, based on information in [2,5][2]. Coordination assistance technology has applications in many areas besides military command and control of course. For example, in an office or concurrent engineering environment, both independent computerized agents (usually controlling access to resources) and intelligent assistants to office workers can exist[25]. In real offices, activity

[1] For example, imagine that your task is to find a new book in a library, and you can do this either before or after the new books are unpacked, sorted, and correctly shelved.

[2] This example was originally presented at Concurrent Engineering '95.

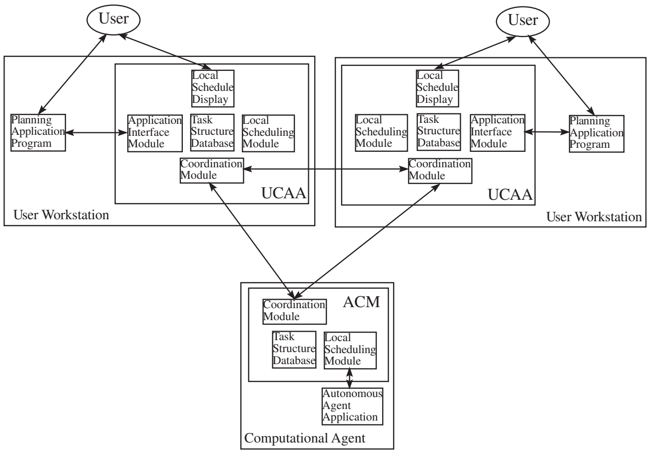

Fig. 1. Architectural overview of a cooperative, mixed human and computational agent system based on User Coordination Assistant Agents (UCAAs) and Agent Coordination Modules (ACMs)

is taking place semi-autonomously, and centralized control and detailed global views are rarely available or socially possible [21,26]. In the intelligent office assistant domain, then, coordination algorithms can be applied to providing guidance to an office worker (for instance, a computer programmer) about how to prioritize tasks ("what are the most important things to do next?"), given known and discovered relationships between one worker's tasks and the tasks of others. Coordination algorithms do not supply a solution for problems of negotiating outcomes or resolving disparate views, but rather try to avoid negative interactions between agent goals or planned actions (avoiding conflicting actions or inter-agent action sequences), and recognize positive or helpful interactions (such as the potential to do another agent a favor, or send some preliminary, predictive information)[33]. Often the coordination process triggers a process of negotiation.

Figure 2 shows a portion of the global task structure for the air campaign planning environment. No one human or computational agent (from here on, referred to simply as 'c-agent') is responsible for all of these tasks, and this global view is never actually constructed—not only would it be expensive to construct in terms of time and communication resources, but it would very likely be both incomplete and out-of date very quickly. Instead, the UCAA that advises each human operator will have a *partially global* view of what's going on (we'll show this in Figures 3 through 4). The complete scenario expressed by Figure 2 concerns the JFACC developing several objectives, identifying a course-of-action for each objective, and then assigning the objective and course-of-action to a planning staff member at their own workstation to develop detailed targeting plans. The development of target sets will in turn involve the work of other staff members, who are working on tasks involving the areas of intelligence, weather, logistics, etc.

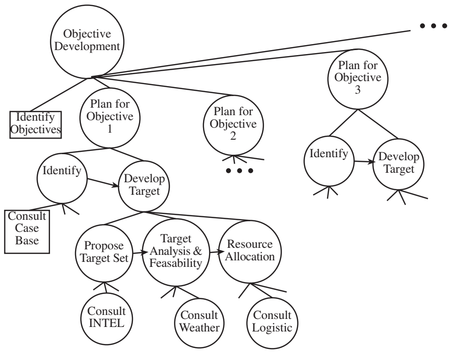

Fig. 2. Part of the global task structure in the Air Campaign Planning environment. Typically, no single agent would ever actually construct such a complete global view.

We'll begin with the JFACC, who is using air campaign planning tools to refine general objectives into the selection of specific objectives, and a course-of-action for achieving the objective (such as disrupting aircraft manufacture). Inherent in this process is the analysis of enemy strengths and vulnerabilities [2]; such analysis is tied to ongoing situation analysis and assessment done remotely and by the JFACC intelligence staff (at remote workstations). Along with the air campaign planning tools, the JFACC has a window on his workstation that displays his current agenda of tasks. For the JFACC, who is basically initiating work, the agenda is simple and unordered. From the point of view of the UCAA, the JFACC's task structure may look something like that in Figure 3, where the JFACC is primarily responsible for the grey tasks. The UCAA is still of great use to the JFACC, however. As the JFACC completes a plan objective and course-of-action specification, he can query the UCAA as to the current status of the work of his planning staff (without bothering them directly)—this information, brought about by communication between the JFACC's UCAA and the UCAAs assigned to each planning staff workstation, is used by the JFACC to make informed decisions about dividing work among his staff members. This is an important point about tools that support group planning—they bring the up-to-date, high-quality information needed to make good human tasking decisions to the user (in this case, the JFACC) when they are needed.[3]

Let us assume that the JFACC has identified three detailed objectives so far, and that the task of target development for objective 1 has been assigned to staff planner A, and objectives 2 and 3 to staff planner B. During the first phase of developing a target set, staff planner A begins by finding the 'Center

[3] It is possible to automate this sort of task partitioning for computational agents.

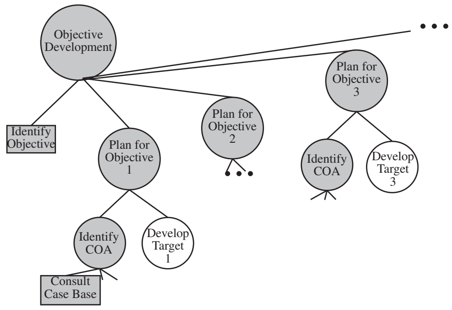

Fig. 3. The local view of the task structure as seen by the JFACC's UCAA. Grey tasks are the JFACC's direct responsibility.

Of Gravity' (COG) of the enemy positions around the objective. Staff planner A's UCAA communicates with the other staff planners' UCAAs to discover if any overlapping work exists—indeed, planner A's UCAA discovers that staff planner B has an objective very close to that of planner A's objective. The COG calculation currently underway by planner A will *facilitate* the COG calculation by planner B. The task structure from the view of planner A's UCAA looks very much like Figure 4 (again, the grey tasks are the responsibility of planner A). The discovery of this relationship was done by the Coordination Modules of the UCAA, using a coordination mechanism we call 'developing non-local views' (we'll discuss coordination mechanisms in more detail in Section 2). The soft relationship as discovered has implications for the scheduling of tasks at planner A's and B's workstations—since B has two tasks to do, and no other reason to order them, B should work on objective 2 first so that B can use the result of planner A's calculations later on in developing targets for objective 3. The Local Scheduling Display for planner B indicates this to him, and the Local Scheduling Display for planner A indicates that B is now depending on the result of his COG calculations (so he shouldn't just delay the task for no good reason). Thus the UCAAs, via their coordination and local scheduling components, have allowed the two staff planners to do their work more efficiently and quickly.

The reader might raise an objection in the last paragraph that the JFACC should have assigned both objectives 1 and 3 to the same staff planner. Perhaps—but there is plenty of reason to *not* do this as well. For example, one objective might have been taking out an aircraft manufacturer, and the second a disruption of communications, and the staff planners may have different areas of expertise. Or perhaps objective 1 is quite hard, long, and involved, and for load balancing the JFACC assigned the other two objectives to staff planner B. In any case, the point is that the technology we are building for use by humans (the UCAA) does not *force* humans into certain actions. At this point in time it would not be realistic to imagine a computer system that could second guess command decisions. Instead the UCAA provides the information needed to make an informed decision, and once it is made, propagates the consequences.

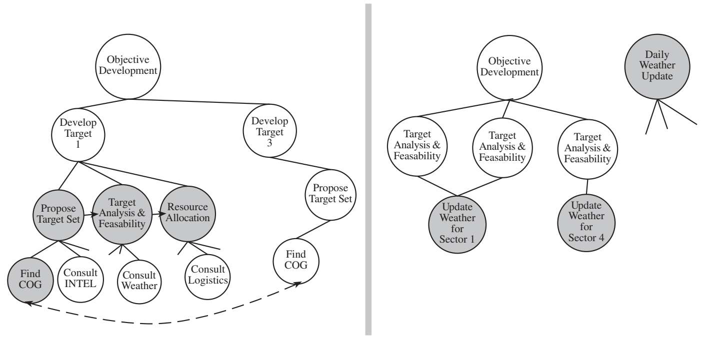

Fig. 4. On the left, the local view of the task structure as seen by the UCAA of member 'A' of the JFACC's planning staff (responsible for the left side grey tasks). On the right, the local view of the task structure as seen by the UCAA of the JFACC staff member in charge of weather forecasting who is responsible for the right side grey tasks.

Finally, let's look at a third member of the JFACC staff, the weather forecaster. From the point of view of planning staff members A and B, the weather forecaster is providing a service; these staff members have the choice of using the current weather forecast for an area, or of requesting a more up-to-date forecast (but which will take time). Perhaps an autonomous weather forecasting computational agent is also available that can give updated forecasts more quickly, but with lower accuracy. The planners can use their UCAAs to see the estimated time to get an updated forecast—such a request is handled by the weather staffer's UCAA and the weather c-agent's ACM. Again, this allows the planner to make an up-to-date, informed decision about whether to go with the out-of-date forecast or wait for the new one.

From the point of view of the weather forecasting agent (human or computational) information about task requests coming in can be used to help the agent schedule their activities. Figure 4 shows the local UCAA's view of the weather staffer's tasks. Notice that the tasks include both those requests from planners for updated forecasts, and standing orders to keep all forecasts not too far out of date. The UCAA's Local Schedule Display will indicate to the user that several tasks are depending on an updated forecast for area 1, and so that might be the best task to do next.

To summarize, our vision is to provide direct support for the goal of concurrent, mixed-initiative planning by groups of humans and computational agents. We will do this by providing an interface that helps the user to schedule their agenda of activities in a sensible way given how those activities interrelate with the activities of other humans and agents in the system. Our support will help make sure tasks get done, and that bad interactions are avoided and positive interactions are taken advantage of. The interface will also enable users to get up-to-date information on the loads of other users and computational agents, so that users who allocate tasks to other users can make better decisions.

2 Approach

Coordination is the process of managing interdependencies between activities. If we view an agent as an entity that has some beliefs about the world and can perform actions, then the coordination problem arises when any or all of the following situations occur: the agent has a *choice* of actions it can take, and that choice affects the agent's performance; the *order* in which actions are carried out affects performance; the *time* at which actions are carried out affects performance. The coordination problem of choosing and temporally ordering actions is made more complex because the agent may only have an incomplete view of the entire task structure of which its actions are a part, the task structure may be changing dynamically, and the agent may be uncertain about the outcomes of its actions. If there are multiple agents in an environment, then when the potential actions of one agent are related to those of another agent, we call the relationship a *coordination relationship*. Each coordination mechanism we describe in Section 2.2 is a response to some coordination relationship.

Developing and applying a domain independent approach to mixed initiative human and computer agent coordination involves developing a language to express agent task structures and potential coordination points (TÆMS); a modular, parameterized, and adaptable approach to the coordination activities themselves (GPGP); and a sophisticated local scheduling mechanism capable of dealing with multiple, dynamically changing evaluation criteria ("design-to-time"[19,18] or "design-to-criteria" [34] scheduling). This section will discuss TÆMS and GPGP. A secondary issue that is also important to keep in mind is the "open systems" nature of our underlying approach. We do not propose to construct a global view of the evolving task structure—most agents will only have a fairly local (or global but abstract) view. Only tasks that have direct relationships need to be understood between agents, and scheduling criteria (or the schedulers themselves) can be different at each agent (and can change over time). Newly developed Coordination mechanisms can be tailored to the agent (human or computational) and task environment at hand.

2.1 A Framework for Modeling Task Environments: TÆMS

In order to construct a general approach to coordination, it is necessary to have an underlying framework that can represent the wide diversity of activities/tasks and the properties and relationships that need to be represented and understood in order to arrive at effective coordination among activity structures of different human or c-agents in a wide variety of domains.

The TÆMS framework (Task Analysis, Environment Modeling, and Simulation) [11] represents coordination problems in a formal, domain-independent way. We have used it to represent coordination problems in distributed sensor networks, hospital patient scheduling, airport resource management, distributed information retrieval, pilot's associate, local area network diagnosis, etc. [9]. For the subject of this paper, there are two unique features of TÆMS. The first is the explicit, quantitative representation of task interrelationships that describe the effect on performance of activity choices and temporal orderings. The second is the representation of task structures at multiple levels of abstraction. The lowest level of abstraction is called an executable *method*, or a basic *action*. A *method* or *action* represents a schedulable entity, such as a blackboard knowledge source instance, a chunk of code and its input data, or a totally-ordered plan that has been recalled and instantiated for a task. A method or action could also be an instance of a human activity at some useful level of detail, for example, "take an X-ray of patient 1's left foot". A *task group* contains all tasks that have explicit computational interrelationships. There may be physical resource relationships between task groups.

A coordination problem instance (called an *episode* $\mathbf{E}$) is defined as a set of task groups, each with a deadline $\mathrm{D}(\mathcal{T})$, such as $\mathbf{E} = \langle \mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_n \rangle$. Figure 5 shows an objective[4] task group and agent A's subjective view of that same task group. Note that no agent ever gains an objective view, but by communicating with other agents, it can expand its subjective view with non-local information, developing a partial global view (see the next section). We use objective-level models in simulation to study the effects on algorithm performance of the presence, absence, or uncertainty associated with certain types of task information. The simulator supports the graphical display of generated task structures, agent actions, and statistical data collection via CLIP[35] in CLOS (the Common Lisp Object System) on various Unix workstations. In real agent systems (Section 3 and 4), of course, only subjective information exists.

The goal of the agent or agents is to maximize the sum of the quality achieved for each task group before its deadline. A task group consists of a set of tasks related to one another by a subtask relationship that forms an acyclic graph (here, a tree). Tasks at the leaves of the tree represent executable *methods*, which are the actual instantiated computations or actions the agent will execute that produce some amount of quality (in the figure, these are shown as boxes). The circles higher up in the tree represent various subtasks involved in the task group, and indicate precisely how quality will accrue depending on what methods are executed and when. The arrows between tasks and/or methods indicate other task interrelationships where the execution of some method will have a positive or negative effect on the quality or duration of another method. The presence of these interrelationships make this an NP-hard scheduling problem; further complicating factors for the local scheduler include the fact that multiple agents are executing related methods, that some methods are redundant (executable at more than one agent), and that the subjective task structure may differ from the real objective structure. The relationships between tasks may also depend on physical resources. This notation and associated semantics are formally defined in [11,9].

A Hospital Scheduling Example We have taken a preliminary look at the application of the described technology to hospital patient scheduling. Let's look

[4] The word 'objective' refers to the fact that this is the true, real structure as viewed by 'nature'.

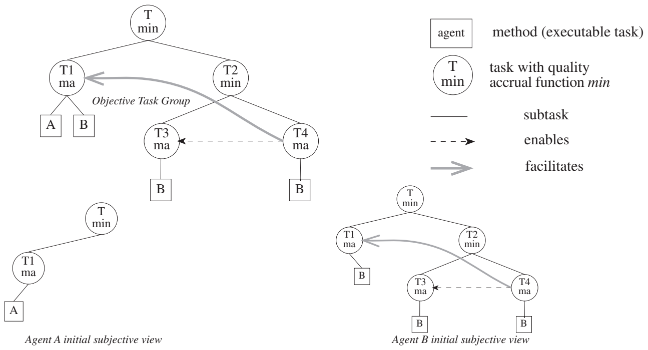

Fig. 5. Agent A and B's subjective views (bottom) of a typical objective task group (top)

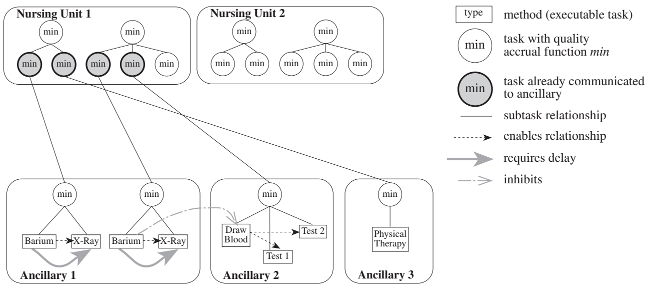

Fig. 6. High-level, subjective task structure for a typical hospital patient scheduling episode. The top task in each hospital ancillary is really the same objective entity as the unit task it is linked to in the diagram.

at a brief example of a task structure model of this environment expressed with TÆMS, our framework for reasoning about multi-agent task environments (see Section 2.1) The following description is from an actual case study [29]:

> *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 (must be performed after), and inhibits (test A's performance invalidates test B's result if A is performed during specified time period relative to B). Note that the unit secretaries (as scheduling agents) try to minimize the patients' stays in the hospital, while the ancillary secretaries (as scheduling agents) try to maximize equipment use (throughput) and minimize setup times.

Figure 6 shows an subjective TÆMS task structure corresponding to an episode in this domain, and the subjective views of the unit and ancillary scheduling agents after four tests have been ordered. Note that quite a bit of detail can be captured in just the 'computational' aspects of the environment—in this case, the tasks use peoples' time, not a computer's. However, TÆMS can model in more detail the physical resources and job shop characteristics of the ancillaries if necessary [9]. Such detail is not necessary for us to analyze the protocols developed by [29], who propose a primary unit-ancillary protocol and a secondary ancillary-ancillary protocol.

In this type of environment, scheduling agents would interact with nurses and each other, using the mechanisms, facilities, and tools suggested here, to assist in the efficient and effective scheduling of patient tests and treatments. Similar applications of our technology occur in manufacturing scheduling applications.

2.2 A Framework for Coordination: GPGP

Many researchers have shown that there is no single best organization or coordination mechanism for all environments. Thus, in order to effectively handle the wide range of possible environments, we have developed an extendable family of coordination mechanisms, called Generalized Partial Global Planning (GPGP), that form a basic set of coordination mechanisms for teams of cooperative computational agents. The important features of this approach include a set of modular coordination components called "mechanisms" (any subset or all of which can be used in response to a particular task environment); a general specification of these mechanisms involving the detection and response to certain abstract coordination relationships in the incoming TÆMS task structure that are not tied to a particular domain; and a separation of the coordination mechanisms from an agent's local scheduler that allows each to better do the job for which it was designed. Each component or mechanism can be added as required in reaction to the environment in which the agents find themselves. An individual algorithm in the family is defined by a particular set of active mechanisms and their associated parameters. In [12] we discuss the interactions between these mechanisms and how to decide when each mechanism should be used, drawing data from simulation experiments of multiple agent teams working in abstract task environments.

The GPGP approach specifies three basic areas of the agent's coordination behavior: how and when to communicate and construct non-local views of the current problem solving situation; how and when to exchange the partial results of problem solving; how and when to make and break *commitments* to other agents about what results will be available and when. The GPGP approach of recognizing and reacting to the characteristics of certain coordination relationships is shared with Von Martial's work on the *favor* relationship [33]. The use of commitments in the GPGP family of algorithms is based on the ideas of many other researchers [3,1,22]. Each agent also has a heuristic local scheduler that decides what actions the agent should take and when, based on its current view of the problem solving situation (including the commitments it has made), and a utility function. The coordination mechanisms supply non-local views of problem solving to the local scheduler, including *what* non-local results will be available locally, and *when* they will be available. The local scheduler creates (and monitors the execution of) schedules that attempt to maximize group quality through both local action and the use of non-local actions (committed to by other agents).

One way to think about this work is that the GPGP approach views coordination as *modulating* local control, not supplanting it—a two level process that makes a clear distinction between coordination behavior and local scheduling. This process occurs via a set of coordination mechanisms that post constraints to the local scheduler about the importance of certain tasks and appropriate times for their initiation and completion. This indirect aspect is extremely important for a system that will integrate both human and computational agents—when dealing with a human agent, the process is necessarily one of providing the correct information at the proper time to the human decision-maker, as opposed to direct control. By having separate modules for coordination and local scheduling, we can also take advantage of advances in real-time scheduling algorithms to produce cooperative distributed problem solving systems that respond to real-time deadlines. We can also take advantage of local schedulers that have a great deal of domain scheduling knowledge already encoded within them. Finally, we can rely on humans as well in making local scheduling decisions.

The way that we specify the family of algorithms in a general, domain-independent way (as responses to certain environmental situations, interactions with a local scheduler, and decisions made by a human or computer) is very important. It leads to the eventual construction of libraries of reusable coordination components that can be chosen (customized) with respect to certain attributes of the target application. It is from this perspective that we feel we can customize our UCAA and ACM to the requirements of the specific application domain. The observation that no single organization or coordination algorithm is 'the best' across environments, problem-solving instances, or even particular situations is a common one in the study of both human organizational theory (especially contingency theory) [24,17,31] and cooperative distributed problem solving [16,14,13,10]. Key features of task environments demonstrated in both these threads of work that lead to different coordination mechanisms include those related to the structure of the environment (the particular kinds and patterns of **interrelationships** or dependencies that occur between tasks) and environmental **uncertainty** (both in the *a priori* structure of any problem-solving episode and in the outcome's of an agent's actions; for example, the presence of both uncertainty and concomitant high variance in a task structure). This makes it important for a general approach to coordination (i.e., one that will be used in many domains) to be appropriately parameterized so that the overhead activities associated with the algorithm, in terms of both communication and computation, can be varied depending upon the characteristics of the application environment.

The Coordination Module The role of the coordination mechanisms is to provide constraints to the local scheduler that allow the local scheduler to construct objectively better schedules. The mechanisms fulfill this role for an agent by (variously) communicating private portions of its task structures, communicating results to fulfill non-local commitments, and making commitments to respond to coordination relationships **between** portions of the task structure controllable by *different* agents (as in the JFACC assigning tasks to his staff example) or **within** portions controllable by *multiple* agents (as in the case of the overlapping tasks in the JFACC example).[5]

The five mechanisms we described in [12] form a basic set that provides similar functionality to Durfee's original partial global planning algorithm [15]. Mechanism 1 exchanges useful private views of task structures; Mechanism 2 communicates results; Mechanism 3 handles redundant methods; Mechanisms 4 and 5 handle hard and soft coordination relationships. More mechanisms can be added, such as a load balancing mechanism. The mechanisms are independent in the sense that they can be used in any combination. If inconsistent constraints are introduced, the local scheduler would return at least one violated constraint in all its schedules, which would be dealt with so as to maximize subjective utility [12]. Since the local scheduler is boundedly rational and satisfices instead of optimizing, it may do this even if constraints are not inconsistent (i.e. it does not search exhaustively).

3 Multi-agent Financial Portfolio Management

One implementation of these ideas has been in a system named WARREN for financial portfolio mangement [32]. WARREN gives a concrete, working implementation of some of the ACM ideas discussed earlier in this paper. The internal architecture of each agent was designed to be compatible with the TÆMS and GPGP approaches discussed above, but for our initial implementation we have replaced the GPGP coordination module with a simpler HTN task reduction planner. The WARREN system is *open*, meaning agents can come and go at any time. A WARREN organization consists of a portfolio interface agent for each user, two task agents for fundemental stock analysis and price-news graphing, a news information agent for Dow-Jones and Clarinet news, several different stock ticker information agents, and two EDGAR information agents assigned to the SEC's electronic archives for quarterly and annual reports. Organizational information agents include a "matchmaker" or yellow-pages agent that helps an information requestor find the appropriate information server in the dynamic, open system. One important requirement was robustness, so that when any

[5] We say a subtree of a task structure is *controllable* by an agent if that agent has at least one executable method in that subtree.

WARREN agent leaves the system (or crashes) the remaining agents reorganize so as to carry on as effectively as possible. We have conducted an analysis of various features (privacy, efficiency, robustness, etc.) for both matchmade and brokered open agent organizations [7].

In particular, "information agents" that are one step above raw data sources have a stylized set of reusable behaviors that make them easy to create [6] [6]. The main function of an information agent and hence its dominant domain-level behaviors are: retrieving information from external information sources in response to *one shot* queries (e.g. "retrieve the current price of IBM stock"); fulfilling requests for *periodic* information (e.g. "give me the price of IBM every 30 minutes"); monitoring external information sources for the occurrence of given information patterns, called *information monitoring* requests, (e.g. "notify me when IBM's price increases by 10% over $80"). An information agent's reusable behaviors are facilitated by its reusable agent architecture, i.e. the domain-independent abstraction of the local infobase schema, and a set of generic software components for knowledge representation, agent control, and interaction with other agents. The generic software components, based on DECAF [12,27], are common to all classes of information agents as well as to the task and interface agents we have built.

The control process for information agents includes, *communication* of planning goals, *planning* to achieve internal or external goals, *scheduling* the actions within these plans, actually carrying out these actions (see Figure 7), and *monitoring action execution*. This entire "metacontrol" sequence (communicate, plan, schedule, monitor) happens at *every* monitoring point (at the very least, upon completion of each executed action). Thus the architecture allows the interleaving of planning and execution, and allows the agent to quickly adapt to new information and unexpected outcomes. In addition, the agent metacontroller has a shutdown and an initialization process. At startup, the agent executes the initialization process which bootstraps the agent by giving it initial goals, i.e. to poll for messages from other agents and to advertise itself with a matchmaker or broker. The shutdown process is executed when the agent either chooses to terminate or receives an error signal. The shutdown process sends messages from the terminating agent to any current customers and the matchmaker or broker informing them of service interruption.

An information agent's external goals, i.e. answering a one-shot query, setting up periodic queries and monitoring information changes are communicated to it in KQML messages originating from other agents. The communication module receives and parses messages, extracts the information goals and passes them to the planner.

3.1 Planning

The agent planning process is based on a hierarchical task network (HTN) planning formalism. It takes as input the agent's current set of goals, the current set

[6] There is always the problem of building a consistant, shared ontology [23,4,20]).

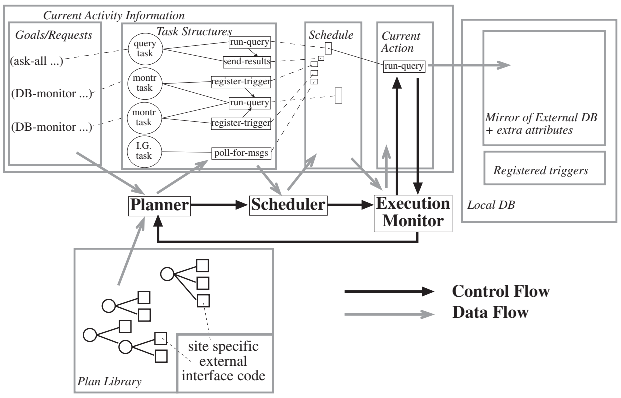

Fig. 7. Overall view of data and control flow in an information agent.

of task structures, and a library of task reduction schemas. A task structure is a partially instantiated task network, including inter-task information flow and other coordination relationships and descriptions of potential task outcomes (See [36] for a complete description of the WARREN task structure representation). Since arrival of goals, planning and execution are on-going and interleaved, the planning process modifies the current set of task structures—by removing tasks, further reducing existing tasks, or instantiating new tasks—to account for any changes to the agent's goals. Task reduction is incremental, and is interleaved with execution.

3.2 Scheduling

The agent scheduling process takes as input the agent's current set of task structures, particularly the set of all primitive actions, and decides which primitive action, if any, is to be executed next. This action is then identified as a fixed intention until it is actually carried out (by the execution component).

3.3 Execution Monitoring

The execution monitoring process takes the agent's next intended action and prepares, monitors, and completes its execution. The execution monitor prepares an action for execution by setting up a context (including the results of previous actions, etc.) for the action. It monitors the action by optionally providing the associated computation-limited resources—for example, the action may be allowed only a certain amount of time and if the action does not complete before that time is up, the computation is interrupted and the action is marked as having failed. Upon completion of an action, results are provided to downstream actions by provision links in the task network, and statistics collected.

3.4 Information Agent Behaviors

An information agent behavior is a particular approach to accomplishing a goal. Behavior instances are represented by a task instance, a set of sub-tasks or primitive action instances, and the information-flow relationships between them. As described above, the task structure for a given goal is instantiated incrementally and dynamically by the planner. The scheduler then determines when individual action instances should be executed, depending on their deadline and on the availablity of required information. As actions are executed and new information is produced, it is routed (as stipulated by information-flow links in the task structure) to other tasks. Thus, the control flow of individual actions within a behavior is determined by information flow. Some of these behaviors are shared in part by agents in WARREN that are *not* information agents—in general these other agents have both reusable and unique behaviors.

Information agent behaviors include:

Advertising: Upon startup, every agent creates an internal goal to advertise itself. An agent advertises itself by sending a matchmaker or broker its infobase schema, which contains all the information needed for describing the information services that the advertising agent can provide.

Message Polling: Message Polling is the simplest information agent behavior. The information agent initialization process asserts a goal for the agent to collect and process incoming KQML messages.

Answering Simple Queries: A simple query is one that is simply applied to the agent's infobase and the results returned to the query-initiator. The query might be a one-shot question or it might be a request for periodic query applications with a given frequency. A task to respond to a simple query consists of two actions: running the query and sending the results to the query-initiator; the running action enables the sending action. The only difference between one-shot and periodic simple queries is whether the constituent actions are themselves periodic.

Information Monitoring: An information monitoring query is one that is interpreted as expressing a condition that when true will trigger the transmission of a selected record or records.

Cloning: Cloning is one of an information agent's possible responses to overloaded conditions. An information agent could recognize via self-reflection that it is becoming overloaded. To do this, it uses a simple model of how its ability to meet new deadlines is related to the characteristics of its current queries and other tasks. It compares this model to a hypothetical situation that describes the effect of adding a new agent. If this evaluation suggests that adding a new agent would be beneficial, the agent removes itself (temporarily) from actively pursuing new queries (by "unadvertising" its services to the matchmaker) and creates a new information agent that is a clone of itself. In this way, the information agent can make a rational meta-control decision about whether or not it should undertake a cloning behavior. A formal presentation of the cloning criteria and the results of empirical evaluation can be found in [8].

4 MADEsmart: Coordination Support for Manufacturing and Design Engineering

We are currently working with a group implementing the UCAA/ACM vision of coordination support as a part of the MADEsmart project at Boeing Helicopters [28]. MADEsmart seeks to partially automate the integrated product teams used to organize design engineers through the use of multi-agent approaches. For example, associated with each human engineer in an integrated product team is a UCAA agent that can interact with that engineer. Other agents, using ACM technology, wrap around existing computationally intensive resources such as composite fiber placement simulations and the COSTADE design cost analysis tool, which uses an existing FORTRAN-based model.

For this project, the core agent achitecture components are being integrated using GBB, a commercial blackboard system. If you look at Figure 7, you can see how the goal, planned task structure, schedule, and domain-dependent data structures could be stored on levels of an agent blackboard, and that the major control components (coordination, planning, scheduling, and the domain actions themselves) could be blackboard knowledge sources and their instances. A graphical task structure specification tool allows programmers to create and edit agent-executable task structures (behaviors), including the flows of information between executable methods / basic actions.

In the initial implementation, the UCAA has little scheduling to do, mostly due to the fact that only one project is being worked on, and the initial task structures have been purposely kept quite spartan. We plan to eventually apply our scheduling technologies to intelligent user interfaces (via the Local Schedule Display in the UCAA). The UCAA will help a user to schedule his or her activities at the workstation and display that schedule (using the Local Schedule Display) in a meaningful and expressive form that can be queried and explained. In most cases, the user will have significant freedom in the ordering of their activities—the purpose of the Local Schedule Display is to make sure that tasks are not forgotten, that time critical or critical enabling tasks are identified to the user, and that facilitating or other soft-related tasks are also identified.

5 Conclusions and Future Work

The most important thing to be clear about the results of our approach is that we are *not* building tools for solving a particular domain problem like crisis management or cost analysis—we are building tools to support the use of domain-level tools by *groups* of people and computational agents to solve large, interrelated problems concurrently. Our results will enable people and c-agents to coordinate their activities so that:

Problems get solved completely. No one drops the ball on an important component for which they are responsible. This is especially important in high pressure situations, and where there has not been enough training.

Problems get solved in a timely manner. Schedules for human and c-agent activities allow for estimates on the completion of important tasks. Humans need to know which tasks on their agendas are time critical. C-agents can use modern resource-bounded reasoning techniques to accomplish tasks by certain deadlines with certain minimal qualities—and to notify others as appropriate when problems develop.

Problems are solved in efficient ways that are dynamically adapted to the current situation. One example is load balancing—more than one agent is capable of solving a subtask, but one can get to it sooner than the other or one has more expertise and is likely to produce a higher quality solution. Another example is exploiting soft task relationships such as where one task facilitates another (but is not necessary), or where one agent could do a 'favor' for another agent (temporarily extending the agent's responsibilities at a low cost). Both positive and negative relationships can exist between tasks, and are usually dealt with through separate mechanisms.

Some important research questions to be answered in this work include the question of how accurate task duration estimates have to be, and how important task monitoring is. We are also looking into automatically learning task duration models, and proper sets of coordination mechanisms [30]. Other important questions include the representation and impact of formal and informal organizational roles when assessing the value of potential schedules.

Our vision of concurrent engineering environments includes intelligent, autonomous computational agents, and agentifying wrappers for existing databases (such as WARREN information agents), that will accept tasks from users or other agents. These tasks could be as 'simple' as one-shot, periodic or database monitoring queries or as complex as the development of initial contingency plans for a certain situation. In a mixed-initiative system we can expect some c-agents to initiate tasks as well (perhaps even to humans). The results of our research will enable these agents to also take advantage of the human user's coordination and scheduling mechanisms so that each agent can intelligently order its tasks, meet deadlines, and take advantage of soft coordination relationships when possible.

References

1. 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.
2. CHECKMATE. Air campaign planning—an approach for the future. White Paper.
3. Philip R. Cohen and Hector J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42(3):213–261, 1990.
4. C. Collet, M.N. Huhns, and W. Shen. Resource integration using a large knowledge base in Carnot. *Computer*, pages 55–62, December 1991.
5. ISX Corporation. ACPT—the air campaign planning tool. White Paper.
6. K. Decker, A. Pannu, K. Sycara, and M. Williamson. Designing behaviors for information agents. In *Proceedings of the 1st Intl. Conf. on Autonomous Agents*, pages 404–413, Marina del Rey, February 1997.

7. K. Decker, K. Sycara, and M. Williamson. Middle-agents for the internet. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, Nagoya, Japan, August 1997.
8. K. Decker, M. Williamson, and K. Sycara. Intelligent adaptive information agents. In *Proceedings of the AAAI-96 Workshop on Intelligent Adaptive Agents*, 1996. AAAI Press Tech Report WS-96-04.
9. Keith S. Decker. *Environment Centered Analysis and Design of Coordination Mechanisms*. PhD thesis, University of Massachusetts, 1995.
10. 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.
11. 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.
12. 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.
13. E. H. Durfee and T. A. Montgomery. Coordination as distributed search in a hierarchical behavior space. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6):1363–1378, November 1991.
14. Edmund H. Durfee, Victor R. Lesser, and Daniel D. Corkill. Coherent cooperation among communicating problem solvers. *IEEE Transactions on Computers*, 36(11):1275–1291, November 1987.
15. E.H. Durfee and V.R. Lesser. Partial global planning: A coordination framework for distributed hypothesis formation. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(5):1167–1183, September 1991.
16. Mark S. Fox. An organizational view of distributed systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(1):70–80, January 1981.
17. J. Galbraith. *Organizational Design*. Addison-Wesley, Reading, MA, 1977.
18. Alan Garvey, Marty Humphrey, and Victor Lesser. Task interdependencies in design-to-time real-time scheduling. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 580–585, Washington, July 1993.
19. Alan Garvey and Victor Lesser. Design-to-time real-time scheduling. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(6):1491–1502, 1993.
20. T.R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. Technical Report KSL-93-4, Knowledge Systems Laboratory, Stanford University, 1993.
21. Carl Hewitt. Offices are open systems. *ACM Transactions on Office Information Systems*, 4(3):271–287, July 1986.
22. N. R. Jennings. Commitments and conventions: The foundation of coordination in multi-agent systems. *The Knowledge Engineering Review*, 8(3):223–250, 1993.
23. W. Kim and J. Seo. Classifying schematic and data heterogeneity in multidatabase systems. *Computer*, pages 12–18, December 1991.
24. Paul Lawrence and Jay Lorsch. *Organization and Environment*. Harvard University Press, Cambridge, MA, 1967.
25. Thomas W. Malone. What is coordination theory? In *Proceedings of the National Science Foundation Coordination Theory Workshop*, February 1988.

26. Sergei Nirenburg and Victor Lesser. Providing intelligent assistance in distributed office environments. In Alan H. Bond and Les Gasser, editors, *Readings in Distributed Artificial Intelligence*, pages 590–598. Morgan Kaufmann, 1988.
27. Tim Oates, M. V. Nagendra Prasad, Victor R. Lesser, and Keith S. Decker. A distributed problem solving approach to cooperative information gathering. In *AAAI Spring Symposium on Information Gathering in Distributed Environments*, Stanford University, March 1995.
28. L. Obrst, M. Woytowitz, D. Rock, S. Lander, K. Gallagher, and K. Decker. Agent-based integrated project teams. Submitted, 1997 ASME Design Engineering and Computers in Engineering Conference; Engineering Information Management Symposium, 1997.
29. 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.
30. 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.
31. Arthur L. Stinchcombe. *Information and Organizations*. University of California Press, Berkeley, CA, 1990.
32. K. Sycara, K. Decker, A. Pannu, M. Williamson, and D. Zeng. Distributed intelligent agents. *IEEE Expert*, 11(6):36–46, December 1996.
33. Frank v. Martial. *Coordinating Plans of Autonomous Agents*. Springer-Verlag, Berlin, 1992. Lecture Notes in Artificial Intelligence no. 610.
34. 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.
35. D.L. Westbrook, S.D. Anderson, D.M. Hart, and P.R. Cohen. Common lisp instrumentation package: User manual. Technical Report 94–26, Department of Computer Science, University of Massachusetts, 1994.
36. M. Williamson, K. Decker, and K. Sycara. Unified information and control flow in hierarchical task networks. In *Proceedings of the AAAI-96 workshop on Theories of Planning, Action, and Control*, 1996.