Environment Centered Analysis and Design of Coordination Mechanisms

Keith S. Decker

UMass CMPSCI Technical Report 95-69
May 1995

Department of Computer Science
University of Massachusetts
Amherst MA 01003-4610

EMAIL: *decker@cs.umass.edu*

This work was supported by DARPA contract N00014-92-J-1698, Office of Naval Research contract N00014-92-J-1450, and NSF contract CDA 8922572. The content of the information does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.

Environment Centered Analysis and Design of Coordination Mechanisms

A Dissertation Presented

by

KEITH S. DECKER

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial
fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 1995

Department of Computer Science

To my parents, Lafeyette J. and Phyllis J. Decker.

ACKNOWLEDGMENTS

When I left General Electric to return to graduate school for my Ph.D., I applied to only one place, UMass, to work with one person, Victor Lesser, on Distributed AI. My deepest thanks to Victor for his guidance, mentoring, and support over the years; Vic pushed me to do more than I thought I could.

I'd like to thank the other members of my committee as well. Paul Cohen supplied invaluable guidance on the statistical analyses in this thesis, as well as reminding me when I failed to remember Tufte's graphical principles. Most importantly, Paul acted as my second mentor here at UMass and his interest in methodology for AI was a catalyst that made the structure of my thesis crystallize for me several years before I completed it. Jack Stankovic was a tough reader who hounded me into avoiding the jargon of DAI and keeping this work open to other computer scientists; he reminded me of how my work interacts with that of traditional distributed computing. Doug Anderton also helped point out my methodological failings and quite a bit of organizational theory literature; he encouraged me to try to open the thesis to readers outside of computer science entirely.

The DAI research community is a friendly and helpful one, and I'd like to thank everyone there for their intellectual stimulus, especially Susan Conrey, Ed Durfee, Eithan Ephrati, Steve Fickas, Piotr Gmytrasiewicz, Mike Huhns, Frank v. Martial, Jeff Rosenschein, and Katia Sycara. A very special thanks goes to Les Gasser, who has continuously reminded me of the social aspects of DAI research and who has always supported that direction in my work.

A very special technical thanks to Alan 'Bart' Garvey, who wrote the Design-To-Time local scheduler (that's part of *his* thesis) used in this thesis—the collaboration has been amazingly fruitful. I'd also like to thank Bart for his gourmet cooking, shipping microbrewery beer and salmon from Seattle, help in brewing (coffee and beer), and his taste in wine.

Many other people have become my friends here and have offered support and encouragement over the years. A special thanks to my extended family, the Westbrooks (David, Teri, Brian, and Josh), who supplied food, washing machine and dryer, gardening advice & supplies, sewing machine, indoor and outdoor recreational activities, and parties. Thanks to the other current and former members of the DIS lab who have been happy to exchange ideas and to put up with proofreading my research papers: Dave Hildum, Marty Humphrey, Dan Neiman, Bob Whitehair, Nagi, Quin (the Alpha Man), Dorothy Mammon, and Sue Lander. I should note that Dave provided the LaTeX style files for this thesis, Marty provided the parallel scheduler I talk about in Chapter 6, and Dann has surely been saddled with reading more of my papers than anyone.

Starting someplace new is always hard, and I made many friends here my first year that have remained friends since. Thanks to Claire Cardie, Lauren Halverson, Alan Kaplan, Sue Mathisen, Doug Niehaus, and Ellen Riloff. Many other friends followed, and I've partaken of their hospitality and/or software experience many times: Scott Anderson, Carla Brodley, Carol Broverman (thanks for the house), Jody Daniels, Tony Hosking, Alice P. Julier, Ruth Kaplan, Joe McCarthy (homebrewer), Zack Rubinstein (Emacs hacker), and David Skalak. Thanks to the Amalgams for giving me something to do over the summers. Thanks to the RCF staff, especially Steve, Terrie, and Glenn.

In a very real way I returned to graduate school to get my Ph.D. due to the support and encouragement of Piero Bonissone, to whom I remain indebted. Special thanks to Jim Aragones for dragging me out of Amherst and back to Schenectady for some fun once in a while.

I should also thank the people at CMU who got me interested in AI, and taught me AI from a cognitive perspective (from which a social perspective on DAI seems a natural one): Herb Simon, Jill Larkin, Elaine Kant, and John Anderson. Thanks to Doc Moore for convincing me to be a math major instead of a physics major. Thanks to Mike and Joan Mintz for also dragging me out of Amherst once in a while.

Finally, a big thanks to my parents for their financial and emotional support through this (very) long process!

ABSTRACT

Environment Centered Analysis and Design of Coordination Mechanisms

May 1995

KEITH S. DECKER

B.S., Carnegie Mellon University
M.S., Rensselaer Polytechnic Institute
Ph.D., University of Massachusetts Amherst

Directed by: Professor Victor R. Lesser
Committee: Professor Paul R. Cohen
Professor John A. Stankovic
Professor Douglas L. Anderton

Coordination, as the act of managing interdependencies between activities, is one of the central research issues in Distributed Artificial Intelligence. Many researchers have shown that there is no single best organization or coordination mechanism for all environments. Problems in coordinating the activities of distributed intelligent agents appear in many domains: the control of distributed sensor networks; multi-agent scheduling of people and/or machines; distributed diagnosis of errors in local-area or telephone networks; concurrent engineering; 'software agents' for information gathering.

The design of coordination mechanisms for groups of computational agents depends in many ways on the agent's task environment. Two such dependencies are on the structure of the tasks and on the uncertainty in the task structures. The task structure includes the scope of the problems facing the agents, the complexity of the choices facing the agents, and the the particular kinds and patterns of interrelationships that occur between tasks. A few examples of environmental uncertainty include uncertainty in the *a priori* structure of any particular problem-solving episode, in the actions of other agents, and in the outcomes of an agent's own actions. These dependencies hold regardless of whether the system comprises just people, computational agents, or a mixture of the two. Designing coordination mechanisms also depends on properties of the agents themselves.

Our thesis is that the design of coordination mechanisms cannot rely on the principled construction of agents alone, but must also rely on the structure and other characteristics of the agents' task environment. For example, the presence of both uncertainty and high variance in a task structure can lead to better performance in coordination algorithms that adapt to each problem-solving episode. Furthermore, the structure and characteristics of an environment can and should be used as the central guide to the design of coordination mechanisms, and thus must be a part of our eventual goal, a comprehensive theory of coordination, partially developed here.

Our approach is to first develop a framework, TÆMS, to directly represent the salient features of a computational task environment. The unique features of TÆMS include that it quantitatively represents complex task interrelationships, and that it divides a task environment model into generative, objective, and subjective levels. We then extend a standard methodology to use the framework and apply it to the first published analysis, explanation, and prediction of agent performance in a distributed sensor network problem. We predict the effect of adding more agents, changing the relative cost of communication and computation, and changing how the agents are organized. Finally, we show how coordination mechanisms can be designed to respond to particular features of the task environment structure by developing the Generalized Partial Global Planning (GPGP) family of algorithms. GPGP is a cooperative (team-oriented) coordination component that is unique because it is built of modular mechanisms that work in conjunction with, but do not replace, a fully functional agent with a local scheduler. GPGP differs from other previous approaches in that it is not tied to a single domain, it allows agent heterogeneity, it exchanges less global information, it communicates at multiple levels of abstraction, and it allows the use of a separate local scheduling component. We prove that GPGP can be adapted to different domains, and learn what its performance is through simulation in conjunction with a heuristic real-time local scheduler and randomly generated abstract task environments.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION: WHY STUDY COORDINATION?

> *In ancient times alchemists believed implicitly in a philosopher's stone which would provide the key to the universe and, in effect, solve all of the problems of mankind. The quest for coordination is in many respects the twentieth century equivalent of the medieval search for the philosopher's stone. If only we can find the right formula for coordination, we can reconcile the irreconcilable, harmonize competing and wholly divergent interests, overcome irrationalities in our government structures, and make hard policy choices to which no one will dissent.*
>
> *— Harold Seidman, Politics, Position, and Power*

Coordination is the process of managing interdependencies between activities [Malone and Crowston, 1991]. This dissertation focuses on the problem of representing these interdependencies in a formal, domain-independent way.

Let us look at the coordination problem in more detail. An agent is some entity that has some knowledge or beliefs about the world (a state) and can perform actions. The problem of coordination occurs when any or all of the following situations occur:

- an agent has a choice in its actions within some task, and the choice affects performance
- the order in which actions are carried out affects performance
- the time at which actions are carried out affects performance

The problem in reality is often more complex. An agent will have difficulty in *choosing* and *temporally ordering* its actions because

- it has an incomplete view of the structure of the task of which the actions are a part
- the task structure is changing dynamically
- the agent is uncertain about the outcomes of its actions

Thus an agent finds itself in a situation (which we call an *episode*) where many of its potential actions are interrelated. An agent does not exist in isolation. An agent is embedded in an environment (or set of environments). An environment implies certain patterns and characteristics of individual episodes within the environment, and of the structures of the tasks within the episodes.

Everything I've said so far applies to a single agent. The final complication is the appearance of multiple agents—each with its own incomplete and possibly changing view of the current episode. When the potential actions of one agent are related to those of another agent (because

there is a choice about what to do or who to do it, or because the order of actions or the time they occur matters) we call the relationship a *coordination relationship*.

For example, in a Distributed Sensor Network (DSN) episode, several computational agents have physically distributed sensors. The area that each agent can sense overlaps with the areas of nearby agents. Vehicles move across these areas, and the agents must work together to track these vehicles. Usually a vehicle will move across the sensed regions of more than one agent, so no one agent will get to sense the entire track. The performance metric for such a system is typically how long it takes the tracking computation to terminate. Each agent has a choice about how to process the data it sees—it might use a fast, low-quality approximation, or a slow high-quality algorithm. If the data is inside an overlapping area, it might be the case that only one agent has to process that data, and for another agent to do so would be redundant and wasteful. It can matter in what order the agents do their tasks. Some types of processing must be done *before* others. For example, low level processing must come before higher level processing of the vehicle tracks. We call this a *hard* predecessor relationship. There are also soft action-ordering relationships in the DSN domain. For example, sometimes an agent can have a faulty sensor that collects noisy signals. These noisy signals take a long time to process. If an agent with a faulty sensor receives information about the type of vehicle it is tracking from an agent without a faulty sensor, it can use this information to filter the noisy data and thus do it's job more quickly. If there are multiple vehicle tracks, then the order in which these tracks are processed can also make a difference. Finally, if the agents are facing a deadline for processing the data, the time at which the actions are carried out can also make a difference. Thus agents in a DSN environment have to make decisions about the choice and temporal ordering of their actions.

This dissertation will demonstrate a framework that can be used to specify the task structure of any computational environment. It will then instantiate an existing methodology (MAD: Modeling, Analysis and Design [Cohen, 1991]) using this framework to analyze a particular computational environment (Distributed Sensor Networks) and predict and verify the performance of two simple coordination algorithms in that environment. Finally, this dissertation will design a family of generic coordination mechanisms for cooperative, soft real-time computational task environments and demonstrate their performance and why a *family* of mechanisms is needed instead of a single static algorithm.

1.1 Representing Task Environments

As I stated at the beginning, this dissertation is about how to represent coordination problems in a formal, domain-independent way. Such a representation should abstract out the details of the domain, and leave the basic coordination problem—the choice and temporal ordering of possible actions. Our solution to this problem is a framework called TÆMS, which stands for Task Analysis, Environment Modeling, and Simulation. TÆMS can be used to specify, reason about, analyze, and simulate any computational environment. The unique features of TÆMS include:

- The explicit, quantitative representation of task interrelationships. Both hard and soft, positive and negative relationships can be represented. When relationships in the environment extend between tasks being worked on by separate agents, we call them *coordination relationships*. Coordination relationships are crucial to the design and analysis of coordination mechanisms. The set of relationships is extensible.

An example of a 'hard' relationship is enables. If some task A enables another task B, then A must be completed before B can begin. An example of a 'soft' relationship is facilitates. If some task A facilitates a task B, then completing A before beginning work on B might cause B to take less time, or cause B to produce a higher-quality result, or both. If A is *not* completed before work on B is started, then B can still be completed, but might take longer or produce a lower quality answer than in the previous case. In other words, completing task A is not necessary for completing task B, but it is helpful.[1]

- The representation of the structure of a problem at multiple levels of abstraction. The highest level of abstraction is called a *task group*, and contains all tasks that have explicit computational interrelationships. A *task* is simply a set of lower-level subtasks and/or executable methods. The components of a task have an explicitly defined effect on the quality of the encompassing task. The lowest level of abstraction is called an executable *method*. An executable *method* 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 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".

- TÆMS makes very few assumptions about what an 'agent' is. TÆMS defines an agent as a locus of subjective belief (or state) and action (executing methods, communicating, and acquiring subjective information about the current problem solving episode). This is important because the study of principled agent construction is a very active area. By separating the notion of agency from the model of task environments, I do not have to subscribe to particular agent architectures (which one would assume will be adapted to the task environment at hand), and I may ask questions about the inherent social nature of the task environment at hand (allowing that the concept of society may arise before the concept of individual agents [Gasser, 1991]). Such a conception is unique among computational approaches. I will adapt Shoham's Agent Oriented Programming terminology when I need to be more specific about an agent's internal architecture (in Chapter 5).

- The representation of the task structure from three different viewpoints. The first view is a *generative* model of the problem solving episodes in an environment—a statistical view of the task structures. The second view is an *objective* view of the actual, real, instantiated task structures that are present in an episode. The third view is the *subjective* view that the agents have of objective reality.

  For example, in the DSN environment, the generative model describes how to generate a DSN episode. The generative model will describe how to generate the number of vehicles that are involved in an episode, and how to generate the objective task structure associated with each vehicle. The resulting objective model describes what tasks are possible at the start of the generated problem solving episode and how those tasks are related. Also generated by the generative model is information about what portions of the generated objective task structure are available to the agents. When an agent

[1] For example, imagine that 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.

executes an 'information-gathering action' the agent then receives a subjective model of some portion of the objective task structure. The 'information-gathering action' here corresponds to the agent requesting data from its sensors.

- TÆMS allows us to clearly specify concepts and subproblems important to multi-agent and AI scheduling approaches. For example, we will discuss the difference between "anytime" and "design-to-time" algorithms in TÆMS. Garvey [Garvey *et al.*, 1993] uses TÆMS to define the concept of a minimum duration schedule that achieves maximum quality.

- This dissertation is about *computational* task environments, where methods are things like blackboard knowledge source instantiations. However, I will also describe extensions of TÆMS to represent physical resource constraints.

The TÆMS representation of an objective task structure is *not* intended as a schedule or plan representation, although it provides much of the information that would go into such uses. TÆMS is not only a formal framework, but also a system for simulating environments: generating random episodes, providing subjective information to the agents, and tracking their performance. The TÆMS simulator is written in portable CLOS (the Common Lisp Object System) and uses CLIP[Westbrook *et al.*, 1994] for data collection.

We validate this framework by building a detailed model of the complex DSN environment of the Distributed Vehicle Monitoring Testbed (DVMT). Our model includes features that represent approximate processing, faulty sensors and other noise sources, low quality solution errors, sensor configuration artifacts, and vehicle tracking phenomena such as training and ghost tracks. Simulations of simplified DSN models show many of the same characteristics as were seen in the DVMT [Durfee *et al.*, 1987]. We also will describe models of many other environments: hospital patient scheduling, a post office problem, airport resource management, multi-agent Internet information gathering, and pilot's associate. Finally, we have validated our framework by allowing others to use it in their work—on design-to-time scheduling, on parallel scheduling, on the diagnosis of errors in local area networks, and in the future to model software engineering activities.

1.2 Analyzing a Distributed Sensor Network Environment

The second major result reported here is a detailed analysis of a simplified DSN environment. The methodology behind this analysis is an instantiation of the MAD (Modeling, Analysis, and Design) methodology [Cohen, 1991], with TÆMS providing the modeling and simulation components. This part of the dissertation returns to the work of Durfee, Lesser, and Corkill [Durfee *et al.*, 1987] that showed that no single coordination algorithm uniformly outperformed the others. This dissertation explains this result, and goes on to predict the performance effects of changing:

- the number of agents

- the physical organization of agents (i.e., the range of their sensors and how much the sensed regions overlap)

- the average number of vehicles in an episode

- the agents' coordination algorithm
- the relative cost of communication and computation

These predictions are verified by simulation.

For example, in Chapter 4 we derive and verify an expression for the time of termination of a set of agents in any arbitrary simple DSN environment that has a static physical organization and coordination algorithm. The total time until termination for an agent receiving an initial data set of size $\hat{S}$ is the time to do local work, combine results from other agents, and build the completed results, plus two communication and information gathering actions:

$$\hat{S}\mathbf{d}_0(\text{VLM}) + (\hat{S} - \hat{N})\mathbf{d}_0(\text{VTM}) + (a-1)\hat{N}\mathbf{d}_0(\text{VTM}) + \hat{N}\mathbf{d}_0(\text{VCM}) + 2\mathbf{d}_0(I) + 2\mathbf{d}_0(C) \qquad (1.1)$$

We can use Eq. 1.1 as a predictor by combining it with the probabilities for the values of $\hat{S}$ and $\hat{N}$. We verify this model using the simulation component of TÆMS.

Our analysis explains another observation that has been made about the DVMT—that the extra overhead of meta-level communication is not always balanced by better performance. This work represents the first detailed analysis of a DSN, and the first quantitative, statistical analysis of any Distributed AI system outside Sen's work on distributed meeting scheduling for two agents [Sen and Durfee, 1994]. This is important because much of the earlier work in this area has been ad hoc, anecdotal, or based on a small number of hand-constructed examples.

1.3 Designing a Family of Coordination Mechanisms

The third major result reported here is the design and evaluation of a family of coordination mechanisms for cooperative computational task environments. We call the collection of resulting algorithms the Generalized Partial Global Planning (GPGP) family of algorithms. GPGP both generalizes and extends Durfee's Partial Global Planning (PGP) algorithm [Durfee and Lesser, 1991]. Our approach has several unique features:

- Each mechanism is defined as a response to certain features in the current subjective task environment. Each mechanism can be removed entirely, or parameterized so that it is only active for some portion of an episode. New mechanisms can be defined; an initial set of five mechanisms is examined.
- GPGP works in conjunction with an existing agent architecture and local scheduler. The experimental results reported here were achieved using a 'design-to-time' soft real-time local scheduler developed by Garvey [Garvey and Lesser, 1993].
- GPGP, unlike PGP, is not tied to a single domain.
- GPGP allows more agent heterogeneity than PGP with respect to agent capabilities.
- GPGP mechanisms in general exchange less information than the PGP algorithm, and the information that GPGP mechanisms exchange can be at different levels of abstraction. PGP agents generally communicate complete schedules at a single, fixed level of abstraction. GPGP mechanisms communicate scheduling commitments to particular tasks, at any convenient level of abstraction.

An example of a GPGP coordination mechanism is one that handles simple method redundancy. If more than one agent has an otherwise equivalent method for accomplishing a task, then an agent that schedules such a method will commit to executing it, and will notify the other agents of its commitment. If more than one agent should happen to commit to a redundant method, the mechanism takes care of retracting all but one of the redundant commitments. The fact that most of the GPGP coordination mechanisms use commitments to other agents as local scheduling constraints is the reason that the GPGP family of algorithms requires cooperative agents. Nothing in TÆMS the underlying task structure representation, requires agents to be cooperative, antagonistic, or simply self-motivated.

In verifying the GPGP family of algorithms, we first show that they duplicate and subsume the behaviors of the PGP algorithm. We look at several other issues:

General Performance: We examined the general performance of the most complex (all mechanisms in place) and least complex (all mechanisms off) members of the GPGP family in comparison to each other, and in comparison to a centralized scheduler reference implementation (as an upper bound). We looked at performance measures such as the total final quality achieved by the system, the amount of work done, the number of deadlines missed, and the termination time. The centralized schedule reference system is not an appropriate solution to the general coordination problem, even for cooperative groups of agents, for several reasons:

- The centralized scheduling agent becomes a possible single point of failure that can cause the entire system to fail (unlike the decentralized GPGP system).
- The centralized scheduling agent requires a complete, global view of the episode—a view that we mentioned earlier is not always easy to achieve. We do not account for any costs in building such a global view in the reference implementation (viewing it as an upper bound on performance). We do not allow dynamic changes in the episodic task structure (which might require rescheduling).
- The centralized reference scheduler uses an *optimal* single-agent schedule as a starting point. Since the problem of scheduling actions in even fairly simple task structures is NP-complete, the optimal scheduler's performance grows exponentially worse with the number of methods to be scheduled. Since the centralized reference scheduler has a global view and schedules all actions at all agents, the size of the centralized problem always grows faster than the size of the scheduling problems at GPGP agents with only partial views. The size of the episodes was kept small so that the centralized reference scheduler could find an optimal schedule in a reasonable amount of run time.

The performance of set of agents using all of the currently defined GPGP coordination mechanisms is good in comparison to the centralized reference system—GPGP agents produce on average 85% of the quality of the centralized upper bound reference solution, and do not miss any more deadlines.

Adding a Mechanism: We demonstrate that the addition of a particular mechanism can improve the system performance.

Family Design Space: We demonstrate the range of performance exhibited by different members of the GPGP algorithm family, obtained by simple parameterization of the individual coordination mechanisms.

Different Environments: We show that different environments require different family members.

Load Balancing: We demonstrate how a new sixth mechanism, a load balancing mechanism, can be defined and integrated. We use this mechanism to show that the costs of using this mechanism are better balanced by performance improvements precisely when there is a large variance in the amount of work each agent would do by default. This result agrees with similar results in the distributed processing community on decentralized load balancing [Mirchandaney *et al.*, 1989].

Computational Organization Design: We recreate a set of experiments done by Burton and Obel [Burton and Obel, 1984] that examined the effects of technical interdependencies and organizational structure on the performance. GPGP team-oriented coordination mechanisms were used to define the organizational (team) structure, and TÆMS task structures defined the problem (as opposed to Burton and Obel's linear programs). We reach the same conclusions as Burton and Obel (that both do have an effect), and argue that one future application for TÆMS is as a tool for computational organization design.

1.4 Placing This Work in a Context

Computer scientists, trying to *design* a coordination mechanism for multiple computational agents, face a problem subtly different from that of the scientists who studied coordination before them: organizational theorists, sociologists, and economists. Sociologists, through observation, want to *explain* how a particular coordination mechanism came into existence, and to describe how it is perceived versus how it really works. Traditional economists *propose* a simple and fairly well-understood coordination mechanism—the market—as the proper mechanism for the idealized rational, utility-maximizing agent. Organizational theorists come from both camps—sociologists and economists—and use *explanations* about the past observations and *proposals* about particular mechanisms to *predict* future behavior. In reality, these distinctions are of course blurred: some computer scientists build sociological descriptions of coordination processes [Hewitt, 1986]; some organizational theorists really attempt the *design* of new coordination mechanisms [Burton and Obel, 1984, Galbraith, 1977]; some computer scientists, both in DAI and in distributed processing, use market coordination mechanisms [Malone *et al.*, 1983, Wellman, 1993], the more complex game theoretic mechanisms [Rosenschein and Genesereth, 1985, Zlotkin and Rosenschein, 1991, Gmytrasiewicz *et al.*, 1991], or team theory [Ho, 1980].

The older schools of thought invariably use either quantitative or qualitative methods of description for the task environment in which the agents under study are immersed. Sociologists need descriptions of the environment to explain "Why?". Neoclassical economists need to make certain assumptions about the environment for their predictions to be correct (and a great deal of work has also been done by economists on what happens when the assumptions are violated). Organizational theorists predict the outcomes of coordination mechanisms given the environment, or show how certain mechanisms are only rational in certain environments.

Furthermore, these descriptions go beyond the enumeration of specific, particular domains to general characteristics of environments.

Artificial Intelligence, growing as it has from the goal of modeling *individual* intelligence, or at least replicating or augmenting it, has focused primarily on representations of individual choice and action. A large effort has gone into describing the principled construction of agents that act rationally and predictably based on their beliefs, desires, intentions, and goals [Cohen and Levesque, 1990, Shoham, 1991]. Fairly recently, researchers concerned with real-world performance have also realized that Simon's criticisms and suggestions about economics [Simon, 1957, March and Simon, 1958, Simon, 1982] also hold for many realistically situated individual agents—perfect rationality is not possible with bounded computation [Horvitz, 1988, Boddy and Dean, 1989, Russell and Zilberstein, 1991, Garvey *et al.*, 1993]. Distributed AI has, for the most part (see Chapter 2), kept the individualistic character of its roots, and focused on the principled construction of individual agents. It hasn't even, so far, really concerned itself with the questions of bounded rationality in real-time problem solving when it comes to the principled construction of individual agents. Worst of all, it has failed yet to bring the environment to center stage in building and analyzing distributed problem solving systems.

In contrast, the organizational science community has since the 60's (e.g. [Lawrence and Lorsch, 1967]) regarded the task environment as a crucial, central variable in explaining complex systems, and a whole branch of research has grown up around it (contingency theory). Representations in this community are rarely formal in nature but rather try to present very rich descriptions using terms such as uncertainty, decomposability, stability, etc. (see Chapter 2).

TÆMS, as a framework to represent coordination problems in a formal, domain-independent way, is unlike any existing representation that is focussed on coordination issues. As a problem representation, it is richer and more expressive than game theory or team theory representations. For example, a typical game or team theory problem statement is concerned with a single decision; a typical TÆMS objective problem solving episode represents the possible outcomes of many sequences of choices (interrelated with one another). TÆMS can represent a game theoretic problem, and we could boil down a single decision made by an agent faced with a TÆMS task structure into a game theoretic problem (if there were no uncertainty involved—see Chapter 2).[2] Because TÆMS is more expressive, we can use it to operationalize some of the rich but informal concepts of organizational science (such as *decomposability* in Section 6.7). Another difference between TÆMS and traditional distributed computing task representations is that TÆMS indicates that not all tasks in an episode need to be done.

To put the second part of this dissertation in context, the analysis of a simple distributed sensor network presented here is the first formal quantitative analysis of a DSN or DAI system, other than Sen's analysis of a two-agent distributed meeting scheduling system (developed at the same time) [Sen and Durfee, 1994]. Our analysis of a DSN system answers several questions, and explains phenomena observed in the work with the Distributed Vehicle Monitoring Testbed (DVMT) such as why different algorithms perform differently in different situations. The work described here moves beyond anecdotal data to design rules for DSN systems.

GPGP, the last major contribution of this dissertation, extends Durfee's work on Partial Global Planning by being domain independent, adding time deadlines, allowing the agents

[2] TÆMS does not say how agents make their decisions. It is perfectly reasonable for an agent to use game-theoretic reasoning processes.

to be more heterogeneous, requiring less communication, and allowing communication at multiple levels of detail. GPGP is a cooperative approach, and thus is different from algorithms that assume the agents act in rational self-interest only. For example, agents usually make decisions with much less *a priori* knowledge of the other agents' utilities than competitive game theory approaches. However, agents using GPGP mechanisms still make decisions in a boundedly rational way—choosing from among schedules in an attempt to maximize the system-wide utility given whatever subjective information they have.

1.4.1 Scope

As I stated at the beginning, coordination is the act of managing interdependencies between activities [Malone and Crowston, 1991]. Coordination behaviors can be divided roughly into *specification* behaviors (creating shared goals), *planning* behaviors (expressing potential sets of tasks or strategies to accomplish goals), and *scheduling* behaviors (assigning tasks to groups or individuals, creating shared plans and schedules, allocating resources, etc.). This work will be primarily concerned with *scheduling* behaviors. Coordination behaviors in turn rest on more basic agent behaviors such as following rules, creating organizations, communicating information, and negotiation or other conflict resolution mechanisms. The agents can be humans, computer programs, or some mixture of the two.

Let us look at an example of these types of behaviors:

Specification. Household robots, business units, and national governments all apply different specification behavior mechanisms to arrive at consensus on more-or-less shared goals. For example, the robots should not run into one another, the business will decide what products to produce, and the government will decide what initiatives to pursue. For the robots, the specification may take place outside of the robots control, in the social fabric in which the robots will be placed. If the robot designers make an arbitrary choice—having the robots always pass one another on the right, for instance—their human owners in England, Australia, or Japan may constantly find themselves in the sidestepping dance common to American tourists walking down the street in these countries. The point is that the specification of passing behavior for the household robot has taken place outside of the robot's control.

Planning. For each shared goal, multiple plans to achieve that goal are possible. These plans may involve multiple agents but are not always laid out explicitly. The robots' behavior may be mostly pure preconceived reaction; the business unit may carefully construct and compare potential plans; the government may embark on multiple, potentially conflicting plans simultaneously.

Scheduling. Finally, planning (or lack thereof) leads to scheduling action—the robots move while integrating the obstacle avoidance behavior with plans to achieve other, private, goals; within the business unit tasks are assigned to people, resources are allocated, and explicit schedules are created; organizations within the government change standard operating procedures and alter decision criteria to incorporate new policies.

All of these classes of behavior—specification, planning, and scheduling—are closely related. Agents plan to meet multiple goals, and schedule actions from multiple plans. Schedulable actions can include the derivation of new goals or plans, either locally or at

other agents. When I say that this thesis will focus on scheduling behaviors I mean that I will not discuss where the goals come from, or how planning occurs (which is itself of course the subject of considerable study in AI). I will however consider the existence of multiple, potentially incompatible goals, and multiple, potentially incompatible plans.

1.4.2 Applications

The work described here was developed with many applications in mind. This section briefly describes some example multi-agent coordination problems in many different domains.

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[Malone, 1988]. In real offices, activity is taking place semi-autonomously, and centralized control and detailed global views are rarely available or socially possible [Hewitt, 1986, Nirenburg and Lesser, 1988]. 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 goals and the goals of others.[3] 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)[v. Martial, 1990]. Often the coordination process triggers a process of negotiation. Similar arguments apply to other domains, such as the concurrent engineering of products or the myriad of scheduling domains (including distributed meeting scheduling [Sen and Durfee, 1994]).

For an example involving computational agents only, imagine a large, heterogeneous local area network and a set of distributed network diagnosis agents assigned to detect and diagnose network communication problems [Sugawara and Lesser, 1993, So and Durfee, 1992]. Now imagine the situation of a low-bandwidth communication link between two subnets. Two agents have a diagnosis method that uses the low-bandwidth link. When a problem occurs, perhaps due to this low-bandwidth link, each (initially uncoordinated) agent may execute its copy of these diagnosis methods at the same time, and the link is saturated—the agents have caused the very problem they are trying to diagnose! Sugawara discusses a solution to this problem where the agents learn coordination rules as they are needed. Later I will discuss the representation of this problem and how such a representation leads to a generalization of Sugawara's solution.

Another example application is agile manufacturing scheduling. Today's manufacturing systems are moving beyond just being 'flexible' (manufacturing many different items) to being 'agile' (adapting production schedules quickly to changes—in orders, production problems, in supplies or equipment) Distributed factory scheduling systems can operate at two levels: between companies (i.e. contractor or supplier relationships, JIT manufacturing, enterprise integration) and within a company. Within a single company, distributed scheduling can still be worthwhile—for example each workcell might schedule its tasks in parallel, and conflicts

[3] Some computer supported cooperative work (CSCW) research has concentrated on the act of the intelligent assistant discovering the goals of the human worker[Broverman *et al.*, 1987, Huff and Lesser, 1988].

that arise between workcells during the day (errors, equipment failures, faulty estimates) might be rescheduled as they happen (scheduling processes can also be developed to be resistant to such errors in the first place). How should workloads be shifted (rescheduled) when a failure occurs? When a customer asks for a delivery date, can a much more accurate date be given by actually adding the potential order to the existing schedule (as a query) rather than giving a seat-of-the-pants estimate? [Sycara *et al.*, 1991, Neiman *et al.*, 1994]

Long-haul telephone switching networks are already sophisticated distributed statistical control applications. The basic problem is how to route calls from, for example, one side of the country to another. When a switch becomes overloaded, neighboring switches need to route calls around the overloaded switch. Current solutions to this problem use statistical techniques to re-route calls and balance the load (based on local statistical information). However sometimes re-routing is not the correct solution. If, for example, the overloaded switch is servicing a national call-in talk show, the rerouted calls that are for the talk show should not be re-routed, but denied instead [Adler *et al.*, 1989]. The implication is that statistical routing could be supplemented with a decision process that takes into account symbolic information about call content, or other types of non-local information.

Distributed delivery tasks are another example application [Sandholm, 1993, Durfee and Montgomery, 1991]. For example (from TRACONET), requests for pick-up and delivery at arbitrary locations come into various regional trucking centers. Given the particular deliveries being made, the number of trucks, etc., often it is worthwhile (because of lower mileage) to exchange jobs with other centers (both centers ending up better off in a distributed search for Pareto-optimality). Such coordination processes can be useful even between different trucking companies (a situation where a centralized scheduler would not be appropriate).

Coordination behaviors in distributed sensor network (DSN) applications like the Distributed Vehicle Monitoring Testbed (DVMT) will be discussed in great detail in this thesis. To state the problem briefly, several agents with physically distributed sensors have the task of tracking vehicles moving through the combined sensed area. Coordination opportunities include avoiding redundant work in areas where the sensors overlap, balancing the processing load, and providing helpful results when some sensors are faulty or the data is open to many possible interpretations.

Distributed information gathering applications might look something like the following: The original user query would be transformed into set of agents, each with their own plans for gathering information, where some plan elements must deal with the coordination of activities and construction of the final query response. Agents could be assigned to concurrently pursue different sources to answer different aspects of the query or to make use of alternative types of sources (e.g., text vs. images) to generate a more comprehensive answer. The agents would proceed to work in a distributed, asynchronous fashion, but there may be a need for coordination among the agents. For example, the results of work by one agent may suggest the need for some of the existing agents to gather additional information from their sources or use alternate sources, or the results might suggest the need for additional agents. The agents must also deal with uncertainties about the availability of sources and the workload associated with the sources, which may require a new division of tasks among the agents [Carver, 1994].

Computational organization design tools use computer modeling of coordination. Example work includes the Virtual Design Team simulator for civil engineering projects [Levitt *et al.*, 1994], designing organizations for decision-making under stress [Lin and Carley, 1993],

the Business Process Handbook project [Malone *et al.*, 1993], computer-based modeling of software engineering processes [Mi and Scacchi, 1990], the ACTION system for re-engineering electronic small-parts manufacturing organizations [Hulthage, 1994], and the work I will describe in this dissertation on modeling computational agent environments and designing new coordination mechanisms.

1.5 Chapter Summary

A coordination problem consists of some environment, the occurrence of episodes within that environment, and the structure of tasks within the episodes. Activities are related to one another so that the choice and temporal ordering of the activities affects some performance criteria. Agents may have incomplete views, task structures may change during problem solving, and agents are uncertain about the outcomes of their actions.

The basic problem which this dissertation addresses is how to state coordination problems in a formal, domain-independent way. Our approach is to first develop a framework, TÆMS, to directly represent the salient features of a computational task environment. The unique features of TÆMS include that it quantitatively represents complex task interrelationships, and it divides a task environment model into generative, objective, and subjective levels. We then extend a standard methodology to use the framework and apply it to the first published analysis, explanation, and prediction of agent performance in a distributed sensor network problem. We predict the effect of adding more agents, changing the relative cost of communication and computation, and changing how the agents are organized. Finally, we show how coordination mechanisms can be designed to respond to particular features of the task environment structure by developing the Generalized Partial Global Planning (GPGP) family of algorithms. GPGP is a cooperative (team-oriented) coordination component that is unique because it is built of modular mechanisms that work in conjunction with, but do not replace, a fully functional agent with a local scheduler. GPGP differs from other previous approaches in that it is not tied to a single domain, it allows agent heterogeneity, it exchanges less global information, it communicates at multiple levels of abstraction, and it allows the use of a separate local scheduling component. We prove that GPGP can be adapted to different domains, and learn what its performance is through simulation in conjunction with a heuristic real-time local scheduler and randomly generated abstract task environments.

The next chapter, Chapter 3, will present and discuss the TÆMS modeling framework in detail. Chapter 4 will present the analysis and verification of a model of a simple distributed sensor network environment, and Chapters 5 and 6 will present and verify the GPGP family of cooperative coordination algorithms. Chapter 2 will discuss other work related to this dissertation. Finally, Chapter 7 summarizes this work and points out the many short- and long-term research pathways that arise from it.

CHAPTER 2

RELATED RESEARCH

> *There is no best way to organize. Any way of organizing is not equally effective.*
>
> *— Jay Galbraith, 1973*

> *Our intent is to show that overly specialized organizational structures allow effective network performance in particular problem-solving situations, but that no such organization is appropriate in all situations.*
>
> *— Durfee, Lesser, and Corkill, 1987*

> *We also expected different dimensions to resolve [coordination] conflicts better in different situations. . . . We also expected each dimension to be applicable at different levels of detail, with varying results.*
>
> *— Durfee and Montgomery, 1991*

The first subsection discusses briefly the reasoning behind the task environment modeling approach. The next two sections discuss the questions of '*What is a coordination behavior?*' and '*How does the task environment affect coordination behavior?*'. These sections will concentrate on the organizational theory literature, because it is probably much less well known to computer scientists. Afterwards I'll return to the Distributed AI conceptual view. General background on Distributed AI has been published by the author in [Decker, 1987]. An introduction to Distributed AI testbeds was published by the author in [Decker, 1994a].

2.1 Task Environments

The reason we have built a framework for task environment modeling is rooted in the desire to produce general theories in AI [Cohen, 1991, Cohen, 1992]. At the very least, our framework provides a featural characterization and a concrete, meaningful language with which to state correlations, causal explanations, and other forms of theories.

The form of a task environment model in our framework is drawn from several sources. First and foremost is our own and others work in the DVMT and similar domain environment simulators [Corkill and Lesser, 1983, Lesser and Corkill, 1983, Durfee and Lesser, 1987, Durfee *et al.*, 1987, Cohen *et al.*, 1989, Decker *et al.*, 1990, Carver *et al.*, 1991, Decker *et al.*, 1991, Decker *et al.*, 1992]. It is from this work that the basic model form—the execution of interrelated tasks—is taken.

One possible form might have been to create a simulator (ala Tileworld [Pollack and Ringuette, 1990]), but it would be very hard to state good featural characterizations using a simulator. The second input to the form of the model is the mathematically formal work in DAI (for example, [Genesereth *et al.*, 1986, Rosenschein and Genesereth, 1985, Malone,

1987, Cohen and Levesque, 1990, Gmytrasiewicz *et al.*, 1991]). There was no reason that the complexities that occur in our earlier DVMT simulation work could not be given clear semantics.

The existing formalisms, however, universally eschew much complexity to allow for optimal analyses to be carried through. For example, Malone [Malone, 1987] formalizes classical organizational and economic coordination devices (i.e., product hierarchies, functional hierarchies, centralized and decentralized markets) and analyzes them using queuing theory. This analysis is oriented mostly toward task assignment (i.e., how much time does it take to assign and execute a task, how much communication is needed, and how vulnerable is the structure to a failure of some component). Tasks are assumed to be independent with exponential arrival times, and all tasks must be completed. The work of Cohen and Levesque (for example, [Cohen and Levesque, 1990]) makes few assumptions about the environment:

> *"...we model individual agents as situated in a dynamic, multi-agent world, as possessing neither complete nor correct beliefs about the world or the other agents, as having changeable goals and failable actions, and as subject to interruption from external events." [Levesque* et al.*, 1990]*

Instead, Cohen and Levesque concentrate on giving formal definitions and semantics to terms such as 'intention', 'commitment' and 'joint intention'—although they "...do not explore how these ideas can be applied in computational systems that reason about action...". In [Genesereth *et al.*, 1986, Rosenschein and Genesereth, 1985, Gmytrasiewicz *et al.*, 1991], computationally tractable decision processes are worked out for particular environmental situations, such as the prisoner's dilemma and other game theoretical environments. Often game theoretical models of environments [Luce and Raiffa, 1958] are unappealing because they assume a great deal of certain, static knowledge about the structure of the environment. Of course this work is still very useful in providing bounds and comparisons to more complex situations.

The second desired characteristic of our task environment modeling framework is, then, to provide clear semantics but not to force the person doing the modeling to simplify things just so that an optimal solution can be found. In this way we hope to build richer and more realistic models—but ones that are still undeniably simplifications of real problems. We will make simplifications to allow analytical solutions to be derived (see the Prisoner's Dilemma example in Section 2.4.5.4 and the DSN analysis in Chapter 4). We will also use simulation techniques when we don't make such simplifications (see Chapter 5).

The third input to the form of a task environment model was the desire to avoid the individualistic agent-centered approaches that characterize most AI (which is probably fine) and DAI (which may not be so fine). The view that what you and I would both agree to call an 'organization' can be described as some sort of 'larger' individual agent is faulty [Allison, 1971] (I'll come back to this in the next subsection). This is the reason the task environment model goes to considerable length to avoid formalizing, say, the mental structure of an agent interacting with the environment. This perhaps stubborn feature of the model allows us to adapt to the individualistic semantics of Cohen, Rosenschein, or Shoham. What I mean by this is that I can use an environment model I build in conjunction with one of these agent architectures to explore how those agents behave in that environment. I will, for example, use a fairly Shohamesque agent architecture to demonstrate GPGP in Chapter 5. Look back at the

quote from Cohen and Levesque: our modeling framework should provide such a dynamic world of incorrect and incomplete beliefs, changeable goals, failable actions, and interruptions. We wish to allow the model to provide:

> *"...a fabric of knowledge, resources, and actions out of which agents actively and flexibly construct and reconstruct themselves by adding and subtracting resources and changing agent-knowledge boundaries. It is the overall collection of problem solving knowledge that is fixed—not the definition of agents." [Gasser and Ishida, 1991]*

This provides a potentially more open system and social perspective than is usual in DAI research [Hewitt, 1991, Gasser, 1991].

A final problem remains with the form of the model that I can only address as a "future direction": that of the mutual construction of agents and their environment. The organization of the world is not wholly dictated either by people individually or by their environments [Agre, 1991, Latour, 1987].

2.1.1 Example Task Environments

For the purposes of discussion and illustration, several of the concepts in this chapter will refer to two task environment simulators developed at the University of Massachusetts: the DVMT and Phoenix.

DVMT.

The Distributed Vehicle Monitoring Testbed (DVMT) [Lesser *et al.*, 1987] simulates a network of vehicle monitoring *nodes* (agents), where each node is a problem solver that analyzes acoustically sensed data to identify, locate, and track patterns of vehicles moving through a two dimensional space. Each problem-solving node is associated with a sensor or sensors. In the simulated domain environment, each vehicle generates sounds (at particular locations and discrete times) that are described by a *grammar* in terms of signal groups, which are then described in terms of low-level, detectable signal frequencies. We will be analyzing an abstract simplification of this problem in Chapter 4.

Phoenix.

The Phoenix firefighting testbed [Cohen *et al.*, 1989] is another domain simulator. Although not primarily a DAI testbed, it does have multiple agents (firebosses and bulldozers) operating semi-autonomously in a highly dynamic environment (a fairly complex fire simulation based on weather conditions, Yellowstone National Park's physical features, and firefighting data of the burn rates of various land types). Some work has been done on coordinating multiple firebosses [Moehlman *et al.*, 1992] through negotiation over bulldozers considered as resources.

2.2 What is Coordination Behavior: The Social View

When beginning our consideration of how the environment affects coordination, I early on came across the following general problem:

> *Consider the difficulties facing the experimenter asking the following question: "What effect does the environment have on learning ability"? This question meets all the criteria of a problem and yet is stated in such a vague way that the investigator could not pinpoint what was to be investigated. The concepts of environment and learning ability are vague. What environmental characteristics? Learning of what?*
>
> *— An Experimental Psychology Textbook*

This problem of conceptual vagueness can be considered yet another reason behind our extensive mathematical modeling of the environment discussed in the previous section, and the cautious set of basic agent behaviors (execution, communication, information gathering). This is also the reason for focusing our causal model on the production of local scheduling constraints (a basic, observable, measurable output that may be based on other behavior such as communication, negotiation, organization, etc.)

In this section I will first discuss how researchers view coordination behavior, and then I will discuss in what way other researchers have proposed that the environment has an effect on such coordination behavior. Here I will cover the views of social/organizational researchers. Until recently, DAI has had a very impoverished view of coordination, often equating coordination behavior merely with the communication of messages, thus it is profitable to be inclusive about other's ideas. I will return to the DAI perspective later in this chapter (Section 2.4).

> *The recurrent problem in sociology is to conceive of corporate organization, and study it, in ways that do not anthropomorphize it and do not reduce it to the behavior of individuals or of human aggregates.*
>
> *— Guy E. Swanson, The Tasks of Sociology*

Several schools of thought in sociology and management science have their own views of coordination behavior: the rational systems school, the natural and institutional schools, the open systems school, the economic systems school. These researchers often separate *normative* (expected or predicted assuming no outside influences) and observed behaviors. The next several sections will briefly summarize the point of view of each school on what is coordination behavior.

2.2.1 What is Coordination Behavior: Rational Systems

In the rational systems view the most important features of organizations are what their specific goals are and what their normative structure is. Organizational activities and interactions are coordinated to achieve these goals; the goals themselves are explicit and include criteria for choosing the best activity. Agents interact explicitly according to the rules of the normative structure, which describes roles and interactions separately from the personal attributes of the agents occupying those roles [Scott, 1987].

The fundamental rational systems normative structure is the *bureaucracy*. The term has a technical sense, apart from its pejorative colloquial use [Perrow, 1986, Weber, 1947]. This normative structure promotes several types of coordination behaviors:

Rules and Regulations: The creation and adherence to rules and regulations is one coordination behavior observed by rational-legal bureaucracies. Rules and regulations apply to individual agents or subgroups. Their purpose is to:

- Justify decisions or actions (especially unpleasant ones)
- Control interactions with the environment

Standardization: The creation and use of standards is a special case of regulations put into place to reduce uncertainty, both within and organization and in its interactions with the environment.

Hierarchy: Hierarchies specify where knowledge, skills and other resources lie, and thus contain communication information. Most importantly, hierarchies outline the nominal power structure — who makes which decisions. Constructing or accessing hierarchies is a coordination behavior.

Specialization: individuals or groups with tightly coupled communication requirements are often *departmentalized*. This functional division is another coordination behavior.

There exists a natural tension between hierarchy and task specialization. A hierarchy groups together routinely related tasks, but in non-routine situations may be a hindrance to coordination of widely separated groups. The modern solution, from the rational open systems view, is to create special task groups that set apart from the routine hierarchy for non-routine situations, transferring the appropriate decision making power to a closer point. If the non-routine situation persists, it may result in a permanent change in the organization. Thus the creation of special task groups is another coordination behavior (although it would be properly classified as a 'rational open system' approach).

2.2.2 What is Coordination Behavior: Natural Systems and Institutions

The natural systems, human relations, and institutional views have in common the fact that they are all reactions to the the rational systems view. They point out that organizations are made up of people, not automatons, and the rational systems view does not take that sufficiently into account.

Of course, for our purposes of studying coordination in distributed problem solving systems, organizational actors *are* (often) automatons! Thus these points-of-view do not provide me with the easy insights that the rational (and rational open) systems views do. I cover these views here for the light they may shed on mixed human-computer coordination behavior. Furthermore, some amount of "irrationality" may in fact facilitate effective problem solving [Lesser 1991, personal communication]. For example, the node skepticism idea [Corkill and Lesser, 1983] advocates indirect organizational structuring, which allows each agent to decide whether to accept its organizational role. A node may choose to be guided more by the proposed organizational structure (to be a "company node") or may decide that its own interests are more important (an "entrepreneurial node"). This freedom is important since the organizational structuring was done with uncertain and incomplete information and therefore may be incorrect. A certain fraction of nodes with each 'personality' may result in a more robust organization in the face of changing environments, an observation that has been made in reference to colonies of honey bees [Reed and Lesser, 1980].

A common tenet of these views is the concept of *structural-functionalism*, i.e., organizational structures can be explained by examining their functions, and certain functions imply certain structures; however [Perrow, 1986]:

> *The explanation for organizational behavior is not primarily in the formal structure of the organization, the announcements of goals and purposes, the outputs of goods and services. It lies largely in the myriad subterranean processes of informal groups, conflicts between groups, recruitment policies, dependencies on outside groups and constituencies, the striving for prestige, community values,the local community power structure, and legal institutions.*

Institutional organization theory [Meyer and Scott, 1983] tries to explain the actions of organizations as attempts to manage their perceived, external image so as to secure support from external sources. Thus an organization will prefer to pattern its structure and operations as a copy of other institutions rather than attempting to maximize its efficiency or some other rational-systems-view measurement. For example, Meyer and Rowan [Meyer and Rowan, 1977] argued that when Congress enacted affirmative action legislation, many organizations created an "Office of Equal Opportunity" as a symbolic response—no genuine response was intended.

A primary coordination behavior observed by natural systems and institutional researchers is the construction and use of *informal channels* [Chisholm, 1989], i.e., communications channels that neither exist in the formal hierarchy, nor were created by formal mechanisms. These communication channels can exist both within an organization and between parts of the organization and the environment.

2.2.3 What is Coordination Behavior: Open Systems

The original conceptions of the rational and natural systems views tended to view organizations in isolation from their environments, with a static set of participants and goals (including the survival of the organization itself). The open systems point of view stresses the environment that the organization operates in — how do participants come to the organization and why do they become a part? How does the organization obtain resources (including physical resources, agents, knowledge, and time) from the environment? Scott summarizes the open systems definition as [Scott, 1987]: "organizations are coalitions of shifting interest groups that develop goals by negotiation; the structure of the coalition, its activities, and its outcomes are strongly influenced by environmental factors."

The open systems view of coordination behaviors includes:

- Negotiation and adoption of shared goals by groups and individuals. This is the most complex coordination behavior, often underlying many of the others mentioned here[1].
- The construction and execution of multi-agent actions (plans).

[1] Negotiating coordination strategies include contracting (of course), bargaining, and 'co-optation'. The last one is coordination by representation. Suppose I have two large CDPS systems that are separate, but have strong mutual coordination relationships in some overlapping area. One coordinating activity is to simply involve a representative from each organization in the decision process of the other — rather than for Agent A to try to transmit all of its plans to B, simply ask A to make a choice from B's palette of choices. A will use it's own information and decision procedure, and B can factor A's decision in with it's own view. Other styles of this strategy can exist between large networks of agents where the agents themselves can be traded or merged into new permanent or temporary organizations.

- The *commitment* behavior defined by Gerson [Gerson, 1976]. A commitment is a dependency, or "side-bet", relating initially unrelated negotiations or potential negotiations and their outcomes to those in the situation at hand. The commitment is a flow or constraint on the flow of resources. For example a schedule is a time commitment, a budget a monetary commitment, loyalty is a commitment of solidarity.
- Power sharing — two mostly independent organizations may share seats on their boards of directors, which is a coordination behavior. Delegation of authority is also in this category of coordination behavior.
- Professionalization — some complex problems are solved by giving complete control to a single agent. This agent both makes decisions and executes tasks.
- Slack — creation of slack in resources (computational ('satisficing'), informational ('information buffers'), or physical ('stockpiling')) is a coordination behavior. It works by *reducing* the need for tight coordination.
- Coding — preprocessing organizational inputs reduces uncertainty and complexity in the environment, which in turn eliminates or simplifies other coordination behaviors.
- Forecasting or prediction is a typical coordination behavior if the environment allows it.
- Growth — especially forward or backward vertical integration [Perrow, 1986]. Again, usurping control over part of the environment can either eliminate or reduce the need for coordination, and so certain types of growth can be viewed as coordination behavior.

2.2.4 What is Coordination Behavior: Economic Systems

The newest school of organizational theorists are, or are influenced by, economists. Thus the driving force behind coordination behaviors of agents is the maximization of their individual utilities. This should not be too surprising, given the game theoretic approaches to DAI [Genesereth *et al.*, 1986, Rosenschein and Genesereth, 1985, Gmytrasiewicz *et al.*, 1991]. While the detractors of these theories have strong arguments [Perrow, 1986], many of the arguments are based on natural systems views and are not as convincing when applied to collectively engineered CDPS systems. The DAI work in this area has focussed on making rational decisions when outcomes will be affected by what multiple agents do—this is really a different problem than the distributed scheduling problem addressed by GPGP.

These economic theories do not postulate new *behaviors*, but rather new explanations for existing behaviors. The first, *principal-agent theory*, explains hierarchical organizational behavior in exactly these terms—self-interested agents maximizing their individual utilities [Moe, 1984, Perrow, 1986]. The second, *transaction costs economics*, takes a more encompassing approach in explaining hierarchical organizational structures as arising from bounded rationality, uncertainty, and other features of the environment. I will explain these relationships in Section 2.3.

This section has briefly discussed a few perspectives on coordination behavior that do not originally come from the AI community. The purpose of this section was to get the reader thinking about the idea that there are many different coordination mechanisms and perspectives on these mechanisms. The next section will try to briefly summarize some non-computational perspectives on the the relationship between the task environment and these different coordination mechanisms. It is not an exhaustive list.

2.3 How Does the Environment Affect Coordination Behavior: The Social View

The social conception of the environment [Scott, 1987] is that no organization exists in isolation, but rather in a complex set of relationships with its environment. The environment constrains the types of participants that can make up an organization and includes what other organizations those participants belong to. The environment is the source of inputs to the organization, the source of *technologies*[2] with which to process those inputs, and the sink for organizational outputs. The things that are goals from the organizational point of view are the specialized functions of the organization from the environment's point of view. The complexities of the environment are reflected in the complexities of the organization's structure. Organizations and their environments are interdependent, and each can influence the other.

In characterizing the environment, one has to decide what level of analysis is appropriate. Perrow [Perrow, 1986] enumerates possible analyses at the individual, group, division, organization, inter-organizational, network, industrial, national, and world levels, to name a few. For our purposes in DAI, individual (i.e., an agent with a virtual single locus of control), organizational, and network analyses seem the most applicable. Network analysis is the process of modeling the control and influence relationships arising from an organization. While the boundaries of a single organization, a hospital for example, are more-or-less clear (there are always boundary problems, however), the network of organizations that make up its environment is large and fuzzy (including other hospitals, federal, state, and local governmental agencies, labor unions, professional societies, support groups, etc.) Constructing a model of a single organization without reference to the network it is part of is an error.

Another way to look at the interactions between units of study, and the levels above and below the units, is to think of the sources of inputs, the markets for outputs, the competitors, and the regulators [Scott, 1987]. One can then characterize the environment as a place where resources are located, both physical and information. An organization can be *dependent* on a physical resource, or have *uncertainty* because it lacks some information. Various sociological studies have shown causal effects between certain variables and organizational dependence and uncertainty. Scott mentions the following (I've added my own examples using the DVMT and Phoenix (Section 2.1.1) which he may or may not agree with):

- Dimensions affecting dependency

  Degree of munificence/scarcity: How easy are the resources to come by? Also called *environmental capacity*. Phoenix has a limited number of bulldozers with which to fight fires, but a potentially unlimited amount of fireline can be built. Time can be considered a resource. The fixed DVMT sensors are limited resources.

  Degree of concentration/dispersion: How are the resources spread out in the environment? DVMT sensors are fixed, and evenly distributed. Vehicles can appear anywhere. Bulldozers can be moved anywhere, at the cost of time lost. Fire can appear almost anywhere.

  Degree of coordination/non-coordination: Are the entities in the environment executing orchestrated or structured actions? For example, in the DVMT, some vehicles

[2] Means for doing work. Machines, knowledge, and skills are *technologies*. This is a sociological term; computer scientists would probably use the term *resources*.

can be part of a single pattern, and act in a coordinated way with one another. By contrast, in Phoenix, multiple fires are uncoordinated (i.e., the fires don't decide on a joint plan to burn an area) and independent.

- Dimensions affecting uncertainty

  Degree of homogeneity/heterogeneity: How similar are the entities in the environment to which the organization must relate? Also *complexity*, *diversity*. Vehicles have noticeable differences. Bulldozers are all the same (although one can easily imagine the addition of helicopters, and different types of bulldozers, etc.). Also, the homo-/hetero- geneity of the terrain can lead to changes in the qualities of fires (see also "degree of stability/variability" below).

  Degree of stability/variability: How quickly, (or, alternately, evenly) are the entities in the environment changing? The DVMT has a very stable environment. The Phoenix environment is more variable when one considers the changing winds and weather.

  Degree of threat/security: Is the organization/entity under study vulnerable to its environment? Will "inefficiencies and errors by an organization result in its demise" [McKelvey, 1982]? The DVMT is secure; The Phoenix organization is secure but the bulldozers are not.

  Degree of interconnectedness/isolation: To how many, and how closely, is the entity under study linked to other entities in the environment in a way that the other entities can affect the one under study? Both the DVMT and Phoenix agents are partially isolated from their environments, but fires can directly affect the fireboss by destroying bulldozers. The *intra-entity* connections, between DVMT agents or between multiple Phoenix firebosses, are very strong.

  Degree of coordination/uncoordination: This dimension effects uncertainty about the environment as well as dependence on environmental resources (that may be controlled by other entities).

Scott notes that these independent variables (uncertainty and dependence) are orthogonal. For example, if the entities in the environment are highly coordinated, it decreases environmental uncertainty (they are more predictable) but may increase dependence (because it reduces the number of independently available resource locations).

Another point to keep in mind when analyzing an environment, which should be clear from the proceeding discussion, is that the analysis of an environment is inextricably linked to ". . . the organization, the cognitive work of its participants and their structure of attention, and its information system" and "the characteristics and goals of the organization [. . .] each [organization] functions in a different *domain*"[Scott, 1987]. Or, in other words, the *environment* is linked to the *agent (organization) architecture* and the *desired agent (organization) behaviors*[3][Cohen *et al.*, 1989]. Scott points out the difference between what the characteristics

[3] One of the reasons that the Partial Global Planning coordination algorithm works is *goal reinforcement* — when a remote goal is equivalent to, or is compatible with, local goals, then the combined goal is rated highly; this happens to be a good thing in the DVMT domain and environment[Decker and Lesser, 1990]. To the extent that one believes goal reinforcement is true in the world at large, the above similarity is encouraging.

of the environment are and how the entities *perceive* those environmental characteristics. Perceptions can be used to predict entity actions or *outputs* (goods, services, problem solutions), but unperceived characteristics can still affect *outcomes*, which are the results of the interaction between outputs and the environment. What agents don't know can hurt (or help) them[Scott, 1987]. Scott (and Perrow) demonstrate that market perceptions held by recording industry executives can predict who will get a recording contract, but not how well the record will sell.

An illustrative example of outcome vs. outputs in the DVMT is the case of the ghost track. A ghost track is a weakly sensed vehicle track caused by sonic reflections from a real vehicle, and occurring nearby. A ghost track is coordinated with the track of the vehicle that it is a ghost of; this fact was true in early DVMT environments. However, the DVMT had no knowledge about ghost tracks; it did not perceive the coordination between ghost tracks and vehicle tracks; it was blind to that characteristic of the environment.

2.3.1 How Does the Environment Affect Coordination Behavior: Contingency Theory

One theme of this dissertation is the development of a method of designing coordination algorithms for CDPS systems. Some very special coordination algorithms have provably good behavior under restrictive conditions [Rosenschein and Genesereth, 1985, Zlotkin and Rosenschein, 1990]. Some algorithms claim much greater generality but have a large number of parameters (PGP has 12 cooperation parameters used in the hill-climbing task reordering algorithm, the meta-level organization chosen, and the time-cushion parameter used in the predictability and responsiveness experiments) or are known or believed not to perform well in certain classes of situations (i.e., heterogeneous, dynamic, and 'real-time' situations [Decker and Lesser, 1990]). In any case, there is to this date no coordination algorithm that is truly general.

If we are interested in designing coordination algorithms for intelligent agents, it only makes sense to look at the design of coordination algorithms for humans. Human organizational design was once (before the '60s) faced with myriad designs that all tried to be good for all situations. It was then noticed that very different organizational designs flourished in industries with different technologies (problem solving methods & knowledge), and in industries where external change took place more quickly or more slowly.

It was clear that there was not a single good organizational structure, but it remained to be shown that the structure mattered. Perhaps one would be as good as another. Lawrence and Lorsch [Lawrence and Lorsch, 1967] built a model (using the aforementioned studies and their own studies) that relates uncertainty in the environment to differentiation and integration in the organizational structure. This model and its elaborations are called *contingency theory*. Lawrence and Lorsch define their independent and dependent variables as follows:

2.3.1.1 Definitions

To apply the definitions below, you should have a clear *level* in mind. An *organization* is made up of *parts*. For example, the organization "U.S. Government" is composed of an "executive branch", "legislative branch", and "judicial branch" (the parts). Obviously parts can also be organizations themselves.

The organization "General Electric" comprises "the service businesses", the "aerospace businesses" and "the core businesses". Note that a level of analysis is not necessarily the same,

in human organizations, as the org chart. There are a lot of reasons for this that I doubt anyone is interested in . . . but the formal org chart can be a starting place.

The organization "DVMT" comprises "DVMT agents". DVMT agents comprise a control component and a domain component. Each of these is composed of knowledge sources. Phoenix is composed of a fireboss, a group of bulldozers, a group of watchtowers, and a group of helicopters (?). Each group comprises individuals.

In the set of studies from which this was derived, there are a set of numerical measures associated with the definitions. I can't see these as being too useful, since they are oriented towards the business world. If this is a useful theory, I should be able to build models of *our* systems with it, but I will need a different operationalization of the variables. For example, I would have tremendous difficulty trying to determine the change in profits the DVMT made before and after the Partial Global Planning mechanisms, but I can easily come up with other operationalizations of performance, like "on the same test suite, the new DVMT found $X\%$ more solutions, was $Y\%$ more confident in them, and did it executing $Z\%$ fewer KSIs."

So I will give the original definitions in order to be concrete, but encourage you to think of how the concepts transfer. The data used to develop this theory came from several environments, but the examples I give are from the plastics environment. An organization in the plastics environment comprises research, applied research, marketing, and production parts. Each part interacts with its own subenvironment in the plastics industry.

Environmental Uncertainty: Environmental uncertainty can be measured by three factors:

1. Clarity or uncertainty associated with inputs or measurements. For example, the environment associated with the research part of organizations in the plastics industry has a higher degree of uncertainty associated with its inputs than any other part of an organization in the plastics industry.

   In the DVMT world, I would say that a DVMT agent with a faulty sensor has high uncertainty associated with its inputs or measurements.

2. Uncertainty associated with causal relationships. For example, in the plastics industry, "the performance characteristics required by the customer are well known, but as far as research is concerned, you might as well be asking for the perfect plastic."

   In DVMT world, there are questions like 'do all ducks attack fish?', etc. There is a great deal of uncertainty associated with the causal and deductive relationships in the DVMT, but it is *the same for each agent*.

3. Time span of definitive feedback after making a decision. In the plastics industry, it may be years before a research program pays off or pans out, but errors in plastics production are usually noticed (and corrected) within a week. Research and production are parts of organizations in the plastics industry.

   In the DVMT I can also measure the amount of time between when a decision is made (i.e., a KS is run) and when I know if it is a correct one. However, for the purposes of Theorem 1.1 below, it is sufficient to note that this feedback time is not significantly *different* for each agent.

Degree of Differentiation: Two organizational parts are considered to be more *differentiated* the more they differ along the following 3 dimensions:

1. Formality or rigidity of their structures (are they unorganized, adaptable, or rigid). For example, the production part of one plastics organization was very formal and rigid because it had a deep hierarchy of managers and frequently reviewed their performance and rarely allowed the lines of authority to be breached. The same plastics organization had a research lab that had only a handful of lab managers to whom all scientists, engineers, and support staff reported to. The research lab encouraged (by decree and monetary incentives) collaboration between staff members. Each DVMT agent has the same internal structure.
2. Time-frame orientations (do they solve problems over the course of minutes, hours, days). In one plastics organization, production and sales were concerned with problems and schedules that often lasted less than a month, while research often planned projects lasting years. Each DVMT agent has the same time-frame orientation.
3. Attention to high-level goals (are they equally interested in the quality of the solution, the solution cost, efficiency, utilization). This was simply a measure of whether the smallest studied parts of a part of an organization actually performed in accordance with the organization's stated goals for that part. The scoop in plastics, for example, was that low-level employees did, in fact, behave consistently with the high-level goals except for research scientists, who seemed to have more of an orientation toward production and sales goals than could be assumed from the stated research goals. Each DVMT agent, by construction, attends to system goals.

A fourth dimension, *interpersonal orientation*, was also used as a sign of differentiation, but I had trouble operationalizing the concept. It was measured on a 2 unit scale: a organizational part was either "task-oriented" (research) or "socially oriented" (marketing).

Required Integration: For a given partitioning of an organization into parts, some amount of collaboration, to achieve unity of effort, will be required between any two parts by the overall environment under which they operate. This can be measured on a binary scale—either the linkage is critical or it is routine. For example, in the environment of the plastics industry, there is a critical link between production and research (manufacturing innovative products), and another between sales and research (finding customers for innovative ideas and finding ways to create what the customers want). The link between sales and production, or sales and applied research, is routine. In a different environment, say the container manufacturing industry, the same parts are not critically linked in the same way. Critical linkages in the container industry occur between production and sales (delivery constraints) and production and research (quality control problems)

The classic DVMT environment places critical linkages between agents whose sensor areas are spatially overlapping, because of the need to (1) reinforce the discovery of a track and (2) look for other parts of a multi-vehicle pattern.

Effective Organization: Effectiveness can be measured by technical performance, for example, a business can be ranked by its change in profits, sales volume, and new products introduced over the past 5 years.

Incarnations of the DVMT have been ranked by quality of solution, number of KSIs executed, and (probably unfortunately) CPU time and memory usage.

The model can then be expressed by the following 3 propositions:

1. In effective organizations, the degree of differentiation between the parts of the organization varies proportionally with the diversity of uncertainty in the environments of the parts.

   In other words, the more different the environments were (as measured by environmental uncertainty) the more different the parts of the organization in those environments were, *for effective organizations.* Poorly performing organizations tended to either force all parts of the organization to remain undifferentiated in the face of environmental differences, or to allow differentiation where the environment did not require it.

   Analyzing at the agent level, the classic DVMT, while its environment is uncertain, is not *diverse* (not different for each part, in this case an agent). The model predicts that DVMT agents should be undifferentiated, and explains this by pointing out that the subenvironment of each individual DVMT agent has an equal amount of uncertainty present (measured by uncertainty of inputs, uncertainty in causal relationships, and time spans of feedback on agent decisions).

2. Effective organizations achieve better integration at the critical environmental linkages than less effective organizations.

   For business organizations, the quality of integration was measured using surveys of the management. For example, "Relations between these two units are: (1) Sound — full unity of effort is achieved . . . (4) Average — sound enough to get by even though there are many problems achieving joint effort . . . (7) Couldn't be worse — bad relations — serious problems exist that are not being solved"

   One way to measure the quality of integration between DVMT agents, off the top of my head, would be to measure the percentage of possible joint solutions that are actually discovered by a pair of spatially adjacent (i.e., critically linked) agents. The model predicts that the DVMT should discover a greater percentage of joint solutions between spatially adjacent agents than between spatially non-adjacent agents.

3. I'll break this into two parts:

   > [A] There is an inverse relationship between integration and differentiation.
   >
   > [B] When parts of an organization are both highly differentiated and highly interdependent it is necessary for the organization to develop more complicated mechanisms for achieving integration between them (if the organization is to be effective).

   The relationship [A] was discovered by pairwise comparison of organizational parts. For each pair, measure the amount of differentiation by the definition above. Now measure the quality of integration actually achieved as described in Theorem 1.2. For all organizations in all environments, the more differentiated the parts were, the lower the quality of integration between the parts.

   The relationship [B] is discovered by looking at all organizations operating in the same environment. The better the measured performance of the organization, the higher the quality of integration. Thus high-performing organizations achieved high quality

integration even when highly differentiated. The rest of this and many other studies goes on to explore the mechanisms that these organizations used to achieve high quality integration between highly differentiated organizational parts.

Part [A] predicts for the DVMT that its undifferentiated agents should all have the same amount of resources devoted to integration.

Part [B] doesn't make a prediction for the DVMT, which is undifferentiated.

These are weak conclusions about the DVMT, and are mostly meant for illustrative purposes. The important point is that Lawrence and Lorsch's work shows that:

1. not only was there no single good structure,
2. but that the choice of structure really did matter.

Galbraith [Galbraith, 1977] further illuminated this model by examining why uncertainty in the environment should be a key independent variable. To Lawrence and Lorsch's correlation he added the following causal link:

> *The greater the amount of uncertainty of the task, the greater the amount of information that has to be processed between decision makers during its execution. If the task is well understood prior to performing it, much of the activity can be preplanned. If it is not understood, then during the actual execution of the task more knowledge is acquired which leads to changes in resource allocations, schedules, and priorities. All these changes require information processing* during *task performance.*

Thus the correlations observed by Lawrence and Lorsch can be explained by strategies to increase the ability of an organization to preplan activities, increase flexibility to adapt in the face of not being able to preplan, and decreasing the level of performance required to solve the problem.

2.3.2 How Does the Environment Affect Coordination Behavior: Information Processing

In newer work [Stinchcombe, 1990], Stinchcombe returns to the structure of the uncertainties facing economic organizations that is the basis of contingency theory (and arguably a progenitor of transaction costs economics, below [Padgett, 1992]). Organizations grow toward, and structure themselves around, sources of information important to them. An actual mechanism is postulated: the resulting structures are not so much a result of self-conscious rational choice, or unintended environmental selection[4], but trial and error learning. The information toward which an organization grows is not so much that which is used to forecast or predict the future, but rather information as *news*, information about particular outcomes that indicate with some certainty how the future is going to be, in some way important to the organization:

> *I propose that information about the uncertain future becomes progressively available* in distinct social locations *[. . .] What resolves the uncertainty of [agents] is the* earliest available information *that will show what direction the [agent] ought to be going because of the way the future of the world is, evidently, turning out [Stinchcombe, 1990].*

[4] Another popular organizational model I have not talked about in detail is that of 'natural selection', or the 'population-ecology model' [Perrow, 1986].

2.3.3 How Does the Environment Affect Coordination Behavior: Principal-Agent Theory and Transaction Costs Economics

Why do hierarchical organizations exist? From the DAI perspective, why aren't contract net-style protocols all we need for DAI coordination problem solving? Both principal-agent theory and transaction cost economics answer this question by denying the benevolent agent assumption [Rosenschein and Genesereth, 1985, Genesereth *et al.*, 1986, Zlotkin and Rosenschein, 1990] (again, from the DAI perspective). We will illustrate this with examples drawn from a hypothetical contract-net style system. These illustrations are drawn from [Williamson, 1975, Moe, 1984, Perrow, 1986, Stinchcombe, 1990], although of course none of these authors are referring to DAI systems!

First, imagine a contract net agent putting a task to bid without the benevolent agent assumption. In such a system there would have to be some utility, or 'reward', for an agent to accept a task—probably the bid specification would indicate the amount of the reward—otherwise why would an agent ever accept a task?. Imagine that there are many agents capable of accepting the task, but that they are not all *equally* capable. The agents examining the bid specification know just how capable they are in accomplishing the specified task, but the contracting agent cannot know these capabilities as well (without spending too many resources). For any given level of reward, some agents that are very capable will find the reward offered insufficient for them to take on the task. Other agents will find the reward generous for their meager capabilities, and thus eagerly accept the contract. Thus a situation referred to by the abstruse term *adverse selection* occurs, where the best agents for a task definitely do not bid on it, while the worst agents definitely do. Such things happen all the time in the real world, for example when hiring employees ("With all my experience and education, I'm worth much more than you are offering"), or when selling insurance ("I can't believe these low rates. They must not have found out about our string of equipment fires," thinks Springfield nuclear plant owner Mr. Burns on *The Simpsons*).

The problem of adverse selection can be mitigated somewhat by designing jobs for the common denominator—you don't have to have the best, just the average for some package of skills. Perrow points out that the adverse selection problem, especially with employment contracts, can run both ways (i.e., the individual doesn't really know exactly what their new job will entail (and thus not *really* how capable they are); how awful their new working conditions will be; how many times grad student fees, suspiciously not covered by their waiver, will triple, etc.)

A similar problem occurs after the contract starts. The task that was just contracted out is very complex—the result will have to be judged on a multiple criteria basis, such as on its certainty, precision, and completeness. Each of these terms will have to be operationalized, and probably watered down to a number in the process. The contracting agent does not have the resources to constantly monitor progress of the other agent, except perhaps for a few progress reports along the way. Since the agent is judged by this set of simpler criteria, rather than the true, hard to specify, original goal, what is to keep the contractor agent from doing as little as possible, orienting its work to these secondary criteria rather than the original goal and then sleeping the rest of the day (or, perhaps for computers, moonlighting with other contracts that will also be done just barely to specification)? The recondite term for this situation is *moral hazard*. In the real world, after getting hired, the employee does as little as possible ('shirking'), and what little they do is oriented toward how they are evaluated by management (think Homer

Simpson); after getting fire insurance, there's no real reason to practice fire safety at the plant since it probably won't burn down anyway.

In both cases, there is uncertainty present—an ' informational asymmetry ' between the agents involved.

The solution, incredibly enough, is capitalism!

— Charles Perrow, Complex Organizations

2.3.3.1 Principal-agent Theory

Principal-agent theory uses moral hazard (shirking) as the basis of the existence of hierarchical organization. Let us assume we have several agents who work together to track vehicles, and they gain some global utility for their results. How do they divide up the rewards? Without the benevolent agent assumption, they cannot trust each other to tell the truth about their individual efforts. "Precisely because of the complex interdependence of tasks" [Moe, 1984] and the processing distance between the sensor inputs and the final tracks, it is hard or impossible to objectively determine each individual agent's contribution. If the agents just split things up equally, each agent will bear its own costs and only get part of the reward—but if the agent shirks, the savings accrue only to it, while the reduced reward will be split among all the agents. Thus it may be only rational for the agents to shirk. Depending on the exact costs of work to each agent and the benefits accrued with and without shirking, the situation can be viewed similarly to a multi-way Prisoner's Dilemma, where the defector(s) cannot be pinpointed.

The principal-agent theoretical solution is for the *agents* to hire a *principal* which will spend its computational resources to monitor the agents, determining their individual efforts and safeguarding against shirking. The principal will allocate rewards to each agent appropriate with their effort, and is given the ability to fire and hire new agents to get rid of shirkers. But how do you keep the monitor from shirking? By giving the principal the excess rewards (residuals) left over after each agent gets its share. The principal agent theorists sternly assert that the resulting arrangement is not any type of 'authority' relationship, but more like a set of spot contracts entered into willingly and rationally by all the parties involved.

This type of analysis can be applied to employee / employer relationships, management / stockholder relationships, and even politician / bureaucrat or citizen / politician (as discussed by Moe, but with the idea of a 'residual' redefined). The criticisms also abound, ranging from simple ones about the exact optimal form for a specific environment (should the agents also get part of the residual? could the principal position be rotated among the agents?), to Perrow's vehement attack (basically, the model suffers from the wildly unrealistic assumptions of neoclassical economics; that agents can always leave the principal without cost, and principals fire and hire agents without cost; that it assumes that not all principals will cheat, bringing the system to a stable point where the principals get everything and the agents little; that authority or unequal possession of power does not effect the model; etc.) Nilakant and Rao [Nilakant and Rao, 1994] argue that by concentrating on adverse selection, moral hazard, and randomness as the primary contributors to uncertainty in performance outcomes, principal-agent theory overlooks two critical sources of uncertainty in outcomes:

- incomplete knowledge about the relationship between effort and outcome

- lack of agreement about the measurement of effort, P-A theory preferring to measure only quantity of effort, and not quality or type of effort.

Nilikant and Rao also argue that principal-agent theory ignores facilitative effort, where an agent does not directly effect the outcome.

> *In the beginning, there were markets.*
>
> *— Oliver Williamson, Markets and Hierarchies*

2.3.3.2 Transaction Costs Economics

Williamson's transaction costs approach [Williamson, 1975] combines microeconomic theory with traditional organizational concerns such as bounded rationality and environmental uncertainty. Again, the question being answered is why hierarchical organizations exist—Williamson answers this by showing the conditions under which the open-ended contractual transactions identified with hierarchical organization are more efficient than short- ('spot') or long-term market transactions (and thus rationally preferred). Under this approach, organizational boundaries are not fixed, but rather arise from linkages between distinct features of the task environment and the agents. The approach has gained much support because of its simple explanations of exactly why and where big, vertically integrated firms will come into existence, and where small firms will operate through markets.

The first of these linkages is between environmental uncertainty/complexity and agent bounded rationality. As we have pointed out several times before, coordination is much simpler without the presence of environmental uncertainty, since complete rational decision making is possible (i.e., cooperation without communication [Genesereth *et al.*, 1986]). Imagine, for example, how Phoenix might have been structured in an environment with an unlimited bulldozing service market if fires were easily predictable—make the certain projection, get the price of a bulldozer from the market, and make the tradeoff computation between price-of-bulldozers and amount-of-land-burned. But of course the problem isn't that simple; the Phoenix planner can not make a certain projection, or even a finite number of possible contingencies to specify in advance, a feature of both the intrinsic complexity of fire growth, the environmental uncertainties (wind speed and direction, for instance), and the bounded rationality induced by the planners' finite computational resources. Thus the Phoenix system becomes vertically integrated, acquiring bulldozers that it can control dynamically as certainties about the environment are revealed, rather than by pre-specifying complex contingencies[5].

The second linkage is between small numbers of agents in the environmental market and opportunism (including intentional lying) on the part of those agents. Imagine Phoenix again with independent, and duplicitous, bulldozers. Imagine also that the central planner gets most of its information about the spread of the fire from the bulldozers (adding informational asymmetry to the problem). A bulldozer could lie about the size of the fire, inflating it so that it gets a bigger share (for less work), but this would not tend to work if there are large numbers of bulldozers, so that inter-bulldozer rivalry will tend to damp out opportunism. Small number of bulldozers won't matter if they do not act opportunistically (in the sense of making self-disbelieved statements). It is the twin features of small numbers and duplicitous

[5] Note how this fits in with Stinchcombe's view of information as *news* that indicates the particular future that is becoming reality.

behavior that causes the market to become inefficient with respect to a hierarchical organization. Furthermore, Williamson points out, markets that are originally large-numbers may become small numbers because of first-mover advantages or experience—witness how companies (or the Department of Defense) become tied to their suppliers. Even if the parties have no information asymmetry but have *incomplete* information, market contract problems can occur when it is hard to judge (or specify unambiguously and completely how to judge) the outcome of the task in question.

Stinchcombe and Williamson both point out that the decision between market and hierarchy is not really a black-and-white one. Market contracts can, and often are, be written to have the effect of organizational hierarchies, blurring the distinctions. Perrow, in his critique of Williamson, points out that the original treatment often downplays the intra-organizational transaction costs—hierarchies do not remove uncertainty (although they may make it easier to respond to), and small-numbers opportunism may occur within a hierarchy (although more powerful monitoring is possible, at some price).

2.3.4 How Does the Environment Affect Coordination Behavior: Social Structural Analysis

Social structural analysis is another approach to understanding the behavior of organizations [Burt, 1982, Pfeffer, 1991]. Organizational structure is one component of social structure. The social structure also contains the patterns across an organization of any important social attribute, such as income, tenure in the organization, or gender. Social structure can also involve the physical environment and the patterns of communications. Social structure theorists have shown that some actions are more influenced by the social structure than by the attributes of the individual agents involved. For example, although individuals prefer to marry into their own ethnic background, groups that have large numbers of mixed ethnicities show correspondingly larger numbers of mixed marriages [Blau and Schwartz, 1984]. Other examples include the effect of the percentage of women in a law school class on the percentage of women who participate more in class [Spangler *et al.*, 1978], and the effect of the size of youth cohorts on the amount of juvenile delinquency [Pfeffer, 1983].

2.3.5 Summary: How Does the Environment Affect Coordination Behavior—The Social View

In this section we described several views of the way the environment affects agent coordination behavior. The most important thing to take from this is the recognition in the organizational community of the importance of environmental characteristics such as task structure (dependencies), and various types of environmental uncertainty—features we will represent in TÆMS and return to in the dissertation many times. Furthermore, these theories should be viewed as reasons why these features need to be represented in any environmental modeling framework. In the next section I will describe the state of the DAI field with respect to coordination behaviors.

This chapter has not included all of organization theory, which has become a much larger field since the early days when the rational systems view ruled. There has been much more fragmentation in the field, and "the old polarities between agency and structure, formal and informal, power and authority no longer apply" [Burrell *et al.*, 1994]. Pfeffer [Pfeffer,

1993] claims that organizational studies has become a low-consensus field, very pluralistic and non-elitist.[6]

2.4 What is Coordination Behavior: DAI Perspective

In their introduction to *Readings in Distributed Artificial Intelligence* [Bond and Gasser, 1988], Bond and Gasser (both familiar with sociology and organizational theory) devote a section to the discussion of coherence and coordination that is notable for its breadth. *Coherence* refers to the evaluation of a system as a unit, including such things as solution quality, efficiency, clarity, and graceful degradation. *Coordination*, on the other hand, "is a property of interaction among some set of agents performing some collective activity."

Bond and Gasser enumerate a set of coordination behaviors exhibited by distributed AI systems:

Organization: This category of behaviors includes forming centralized, hierarchical decision-making or authority structures, markets (such as Malone's work and the Contract Net [Davis and Smith, 1983, Malone, 1987]), and structured communities (such as the Scientific Community Metaphor [Kornfeld and Hewitt, 1981]).

Increased Localization: These behaviors are oriented towards decreasing the interactions necessary between agents, and includes *specialization* (narrowing, constraining, reducing, or focusing an agent's responsibilities) and *reducing dependencies* between agents. See also *specialization* in reference to rational systems (Section 2.2.1), and *professionalization*, *slack*, and *growth* in reference to open systems (Section 2.2.3).

Increased Local Capability: Activities involved in making better local decisions can be considered coordination behaviors. This is a driving note in much of Lesser and Durfee's work [Durfee *et al.*, 1987, Durfee *et al.*, 1989]—increased local capabilities (especially better self-realization of an agents plans and problem-solving abilities) can *potentially* lead to increased coherence (through increased coordination). In other words, there is a synergy that can occur where agents with better self-knowledge can coordinate themselves more effectively. Thus the local planning mechanism in the Classic DVMT could be considered to exhibit coordination behavior when it constructs a local node plan.

Planning: Planning is often a coordination behavior (sometimes coordination with elements of the external environment). Bond and Gasser break it into two finer gradations — *multi-agent planning*, where one agent plans for several (for example, Georgeff's work [Georgeff, 1983, Georgeff, 1984]), and *distributed planning*, where plans are constructed mutually by the agents (for example, PGP [Durfee and Lesser, 1987], distributed NOAH [Corkill, 1979], or Ephrati's work [Ephrati and Rosenschein, 1994]). See also planning in reference to open systems, Section 2.2.3. The only work on taking advantage of *positive* plan interactions is von Martial's work on exploiting the *favor* relationship [v. Martial, 1990, v. Martial, 1992].

Increasing Contextual Awareness: Another coordination behavior is the exchange of information regarding agent roles and responsibilities (*network views*, such as in PGP). This

[6] And thus perhaps not as influential as it could be [Burrell *et al.*, 1994].

is often followed by behaviors such as creating models of other agents (their plans, beliefs, knowledge, or all three) and predicting their actions [Genesereth *et al.*, 1986, Halpern, 1986, Wesson *et al.*, 1981]. This is related to communicating abstractions and meta-level information, below.

Managing Communication: Another coordination behavior is observed when agents carefully manage their communication behavior. This management can be in terms of relevance, timeliness, and completeness [Durfee *et al.*, 1987], expectation driven communication [Decker, 1987], or discriminatory or diagnostic content [Wesson *et al.*, 1981, Carver *et al.*, 1991].

Managing Resource Use: Agents that manage their use of resources (usually consumable or otherwise limited resources), including time, are engaging in a coordination behavior. Managing resources can include creating 'slack' and various relaxation and least commitment strategies [Fox, 1981, Chandrasekaran, 1981, Cammarata *et al.*, 1983]. See also *slack* in reference to open systems, Section 2.2.3, and *resource dependency* in reference to the social concept of environments, Section 2.3.

Managing Uncertainty: There are many kinds of uncertainty and many behaviors in reaction to them. Some aspects of uncertainty in the environment seem to impact on agent and organizational coordination behaviors (see Section 2.3 and Section 2.3.1). Fox [Fox, 1981] discusses how various types of uncertainty (informational uncertainty, algorithmic uncertainty, environmental uncertainty, behavioral uncertainty) affect organizational structures, according to contingency theory [Galbraith, 1977] and transaction costs economics[Williamson, 1975, Moe, 1984, Perrow, 1986]. Approximate processing can be viewed as a coordination behavior when it is used to create slack (processing) resources [Lesser *et al.*, 1988, Decker *et al.*, 1990].

Pluralism: Organizational behavior where multiple agents or groups attack the same problem from different viewpoints are exhibiting *pluralistic* behavior, another form of coordination behavior. The FA/C paradigm is cited as an example [Lesser and Corkill, 1981]. Bond and Gasser also consider negotiation from multiple conflicting viewpoints to arrive at a shared viewpoint or temporary solution *due process* coordination behavior [Hewitt, 1986].

Abstraction and Metalevels: The idea here is that forming and communicating abstractions, or exchanging meta-information about the state of coordination processes, are both important coordination behaviors. These ideas are most clearly embodied in the PGP line of research, where the exchange of meta-level information about the schedules of remote node was used, for example, to avoid redundant work [Durfee, 1987]. The work that has followed this has emphasized the use of abstraction to reduce communication and detect activity interactions [Decker and Lesser, 1992, Durfee and Montgomery, 1990].

Adaptation: Self-organization of multi-agent systems is another coordination behavior. The Contract Net, and other related market-based organizations, represents one style of self-organization as agents create temporary organizational structures through contracting in response to incoming tasks. A very different style is used by Gasser and Ishida [Gasser

and Ishida, 1991] where agents themselves are created and destroyed, and knowledge is moved between agents, in response to a changing task load.

This has been a brief description of where work was when my work started. In the rest of this section I will describe some especially relevant pieces of DAI work in more detail.

2.4.1 Malone's Organizational Models

Communication between nodes was the only coordination behavior measured by Malone [Malone, 1987, Malone and Crowston, 1991]. It was measured as the amount of communication between individuals needed in order to assign tasks. It was a fixed constant for some of the organizations he studied (product hierarchies and centralized markets (modeled the same as functional hierarchies)), and proportional to the number of agents for a decentralized market. Malone compared organizational structures with one another, keeping anything that might be considered an environmental factor fixed. Malone's purpose was not in assessing the impact of the environment on coordination, but on assessing the impact of various organizational structures — what I would call coordination algorithms — on the production, coordination (communication), and vulnerability costs. The systems Malone modeled exhibited other coordination behaviors — they used hierarchies, and took part in negotiations — but these behaviors were not modeled.

2.4.2 Partial Global Planning

Partial global planning [Durfee and Lesser, 1987, Durfee and Lesser, 1989] was developed as a distributed control technique to insure coherent network problem solving behavior. It is a flexible approach to coordination that does not assume any particular distribution of subproblems, expertise, or other resources, but instead lets nodes coordinate in response to the current situation. Each node can represent and reason about the actions and interactions of groups of nodes and how they affect local activities. These representations are called partial global plans (PGPs) because they specify how different *parts* of the network *plan* to achieve more *global* goals. Each node can maintain its own set of PGPs that it may use independently and asynchronously to coordinate its activities.

Of all the behaviors that this system exhibited, four of them can be construed as coordination behaviors by Malone's definition:

- It avoids redundant work among nodes by noticing interactions among the different local plans. Specifically, it notices when two node plans have identical intermediate goals, i.e., when they are working on the same time region. This occurs in the DVMT because in the interests of reliability nodes have overlapping sensors. *(subdividing goals into tasks, assigning tasks to groups or individuals, communication of information)*
- It schedules the generation of partial results so that they are transmitted to other nodes and assist them at the correct time. To do this it uses the estimates of the times that activities will take and the inferred relation that if node A estimates that it will take less time than node B to complete an intermediate goal, and the goals are spatially near, that node A can provide predictive information to node B. *(assigning tasks to individuals, communication of information, (computational) resource allocation)*

- It allocates excess tasks from overloaded nodes to idle nodes. Node plans provide the information needed to determine if a node is overburdened or underutilized. A node is underutilized if it is either idle or participates in only low-rated PGPs. A node is overburdened if its estimated completion time of a subgoal of goal G is much later than the completion time of all the other subgoals of G [Durfee and Lesser, 1989]. *(assigning tasks to groups or individuals, resource allocation)*
- It assumes that a goal is more likely to be correct if it is compatible with goals at other nodes. In the DVMT task, a goal represented a processing task to ascertain whether a vehicle was moving in a region r at time t. This goal could, in fact, be wrong — based on noise or erroneous sensor data that was the basis for the preliminary task analysis that generated the goal. Nodes choose local plans to work on based on the highly rated PGPs they have received. Thus, if the intermediate goals of a node become part of a PGP, then they are worked on before other intermediate goals in other local plans the node may have (even though the node may have rated those local plans higher in its local view). *(assigning tasks to groups or individuals, conflict resolution between agents)*

In Durfee's thesis [Durfee, 1987], the partial global planning mechanisms are claimed to exhibit most of the following coordination behaviors:

- Form subsolutions in parallel
- Form solution construction graphs (that show when partial results are to be delivered and to whom)
- Exchange predictive information
- Assign important tasks to multiple agents
- Exchange tasks between agents
- Avoid redundant activities
- Verify other agent's results
- Generate independent work
- Selectively exchange messages

As I will discuss in Section 5.1.1, the work on Generalized partial Global Planning (GPGP) that I will describe in Chapters 5 and 6 extends the PGP ideas. It does this by allowing more agent heterogeneity, the exchange of more truly *partially global* information at multiple levels of abstraction, the use of separate scheduling algorithms, etc. We will return to Partial Global Planning in more detail when we introduce GPGP in Section 5.1.

2.4.3 Hierarchical Behavior Space

Early in this dissertation, I asserted that coordination behavior could be divided into three general areas: specification (creating shared goals, instantiating goals into tasks and subdividing them, standardization, etc.), planning, and scheduling. In more recent work, Durfee and Montgomery [Durfee and Montgomery, 1990] have worked on the specification side of things, developing a set of six hierarchies—who, what, when, where, how, and why—for specifying agent behaviors. This rather arbitrary but nonetheless useful hierarchical standardization

mechanism allows goals to be specified, compared, and subdivided by potentially heterogeneous agents. Durfee and Montgomery also present a 'protocol' (algorithm) for using this information for scheduling coordination, however the mechanism is very general (agents communicate to avoid bad interactions and take advantage of positive interactions) and little work has been done on prescribing exactly what is good behavior in a given environment (see the quote in Section 2.3.1). More recent work has shown that using abstraction does in fact result in less communication [Montgomery and Durfee, 1992]. In the chapter on GPGP (Chapter 5) we will discuss several mechanisms that are much more detailed, and can be used together to construct an extendable family of coordination algorithms. One of those mechanisms (non-local view) tries to minimize communication; we have also argued elsewhere (and at about the same time frame) that such a mechanism can be implemented in a hierarchical manner[Decker and Lesser, 1992] to reduce communication.

2.4.4 Distributed AI and the Concept of Agency

The conception of agency in DAI has a different scope and direction from that of principal-agent theory (see Section 2.3.3). The primary theoretical focus has always been so called 'BDI' (Beliefs, Desires, and Intentions) architectures, probably because of the long history of mathematical logics of knowledge and belief extending back far before AI or modern computer science (Halpern links it to the Greek study of epistemology ('the study of knowledge') [Halpern, 1986]). A large amount of this work is not computational (e.g., Cohen and Levesque's comment from the start of this chapter that they ". . . do not explore how these ideas [of intention and joint intention] can be applied in computational systems . . . " [Levesque *et al.*, 1990]). Informally, BDI architectures state exactly how an agent chooses its actions based on its current set of beliefs and its current set of desires (or goals). Intentions are used to link desires or goals (or plans to achieve goals) to actions.

A good example of a computational expression of these ideas is Shoham's AGENT-0 [Shoham, 1991]. It is not the only such expression (see for example at least [Halpern, 1986, Genesereth *et al.*, 1986, Rosenschein and Genesereth, 1985, Cohen and Levesque, 1990, Zlotkin and Rosenschein, 1990, Gmytrasiewicz *et al.*, 1991, Shoham, 1991, Jennings, 1993]), but it is a clear one. The construction of new *agent* architectures is *not* the subject of this dissertation; this example is just one to use as a reference point when I talk about how agents interact with their environment later in the document. It is also useful to compare this with the agent architecture used by the GPGP algorithm in Chapter 5.

Shoham's AGENT-0 is really a simple interpreter based on the more general idea of *agent-oriented programming* (AOP). AOP specializes the ideas of object-oriented programming (in the Actor sense) so that objects become agents that have a formally defined mental state (in this case, beliefs, commitments, and capabilities rather than beliefs, desires, and intentions) and that send messages to one another with types inspired by speech act theory (inform, request, offer, etc.). Commitments are a type of intention, and desires or goals come about from the speech acts themselves (i.e., requests from other agents). AGENT-0 as a first pass explicitly excludes more complex intentional stances, plans, or goals and desires.

An agent program consists of some initialization and a set of unordered commitment rules. These commitment rules refer to the 'current' mental state of the agent (including current incoming messages from other agents). The result of a commitment rule can be a commitment

to action at any future point in time. The underlying interpreter makes sure that the agents honor their previous commitments.

The interpreter has only these two cycles: process the incoming messages (update local beliefs and then update the commitments by running the commitment rules), and carrying out the commitments already made for the current time. Information from other agents is marked as being the beliefs of these other agents (thus achieving common belief given perfect communication media); if it inconsistent with old beliefs then the old ones are removed (AGENT-0 allows only simple single facts and their negations, allowing this to happen fairly quickly). Rather than institute belief revision, AGENT-0 programs note important preconditions as part of the *capability* clause of a commitment (so retracting a belief can in fact cause a commitment to be no longer carried out because the agent is no longer 'capable' of doing it). Such a retraction is always communicated to other agents.

2.4.5 Distributed AI and Distributed Processing

The problems and techniques of Distributed AI in purely computational systems are somewhat related to those of the distributed computing community [Lampson *et al.*, 1981, Paker and Verjus, 1983, v. Bochmann, 1983, Chambers *et al.*, 1984, Stankovic, 1984a]. The differences arise from several sources: focus, approach, and most importantly scope. Much distributed processing research is *focused* on the problems of data consistency and transaction processing that are not the focus of DAI research because DAI research avoids the creation of consistent global views. Thus one would never build a DAI system for handling bank transactions, but one might build a DAI application to search for instances of fraud on top of such a system. Many distributed processing *approaches* to control and scheduling issues use centralized solutions—perfectly reasonable and efficient solutions in systems where the bottleneck is either not a problem or can be mitigated by special hardware. Many distributed scheduling approaches have assumed that tasks are computationally independent; newer scheduling work admits the scheduling of task groups with precedence relationships (i.e., Spring [Zhao *et al.*, 1987, Stankovic and Ramamritham, 1987]). Some techniques, such as contracting, have been used (independently) by researchers associated with both DAI and distributed computing systems [Malone *et al.*, 1983, Malone *et al.*, 1986]. Most of all, the *scope* of distributed processing work is different; often it can provide a structure for the efficient implementation of DAI systems. In Section 5.7 I will briefly discuss an interface between scheduling and decision making applicable even to hard real-time scheduling; also see [Garvey and Lesser, 1994, Garvey *et al.*, 1994].

2.4.5.1 Decentralized Control Theory

Classical and modern control theory is oriented mostly toward the control of continuous physical processes. A single controller (agent) observes outputs from the environment (plant) and sets inputs that along with the current environment state determine the new outputs. Since control theory is the mathematical basis of most engineered systems, a large body of work exists. One introduction for computer scientists is Dean and Wellman [Dean and Wellman, 1991]. Control theory has been used in AI control for continuous systems like robots. The types of environments that we will model in this dissertation are in general not continuous but discrete; less work has been done in discrete control theory. In this dissertation we are

interested in control in terms of choosing what activities to do and when to do them, which is on the surface different from varying continuous plant input variables like the direction of a robot's travel or the temperature of a curing process. Of course the extremely general nature of control theory allows it to express even these types of problems, but few of the standard control theoretic solution techniques are available. For example, the discrete task environments with complex interrelationships that we will model in this dissertation do not usually satisfy the superposition and homogeneity properties required for the system to be linear. Another difference is that the models we will discuss in this dissertation are more highly structured than dynamical control theoretic models. Although it is far outside the scope of this dissertation, it would be easier to develop automated tools for building TÆMS models than arbitrary dynamical models of computer programs (as opposed to physical systems).

Extensions to control theory include stochastic control, where there is uncertainty in the environment's response to inputs and uncertainty in the controller's (agent's) observations of the environmental outputs. The usual control theoretic approach to observational uncertainty is the Kalman filter approach. This approach can be used to map from observations and a set of linear dynamical (control theoretic) models to a choice of one of these models that is the 'best' (minimum mean-square error). It assumes evenly distributed Gaussian noise as it's model of uncertainty, which does not match all of the uncertainties in the systems we model—for example, one would probably assume that the error in measuring durations of executable methods would decrease over time.

Other extensions include decentralized control theory ([Šiljak, 1991], for example), where there are multiple controllers (and plants). This situation is of course again somewhat closer to the multi-agent systems described in this dissertation, although still focussed on continuous linear systems. A final important point to make about the applicability of control theory is that some computer algorithms are currently being developed in the AI community ("anytime algorithms" [Boddy and Dean, 1989]) and systems communities [Liu *et al.*, 1991] that have continuous response characteristics, and thus some control theoretic techniques can be adapted for them.

2.4.5.2 Team Theory

Team theory [Ho, 1980] is a formalization of a simple distributed control decision problem. Stated informally, this problem is to find a decision rule for each of a set of agents that tells each agent how to set each agent's own 'control' given some set of potentially uncertain observations. Furthermore, the decision rules should minimize the expected value of a predetermined loss criteria based on what the agents do and the true state of nature.

The team theory formalization contains five components [Stankovic, 1984a]:

1. The state of nature: a vector of random variables with given distributions that represent all the uncertainties that have some bearing on the problem
2. A set of observations on the state-of-nature vector for each agent, based on some function of the true state-of-nature.
3. A set of decision variables or controls, one per decision maker
4. A decision rule (strategy) for each agent which chooses a value for an agent's decision variable given the agent's observation vector.

5. A loss (or payoff) function based on the values of the agent's decision variables and the true state of nature.

This fairly simple, static decision problem (i.e. choosing the decision rule for each agent so as to minimize the expected loss) is in general intractable. Some comparisons to TÆMS are possible. The state of nature vector is similar to the TÆMS objective level representation of a problem solving episode; however TÆMS enforces considerable structure on the representation (i.e., task durations, qualities, and interrelationships). In TÆMS the objective representation changes over time as the agents make decisions and take action—it is neither static like team theory nor does the current state of nature come about only externally. The team theory "set of observations" is somewhat similar to the TÆMS subjective level representation, except for the addition of considerable structure as I have just indicated. As noted by Stankovic [Stankovic, 1984a], the set of observations made (subjective information) may in fact be a function of both the true state-of-nature and the actions of other agents. The decision rules (control strategies) in TÆMS are not about how to set some control variable, but rather are about what actions to execute and when. The control strategies (like GPGP together with Garvey's DTT scheduler) are usually dynamic as well.

2.4.5.3 Distributed OS Job Scheduling

Within the realm of distributed computer systems, there has been a lot of research into distributed control, but much of it is not very closely related to the types of problems discussed in this dissertation. For example, work on static, a priori decomposition or task allocation techniques for assigning tasks to processors will often not be applicable because there is too much uncertainty in the structure of the task, or because there is no central authority who can make such decisions (in the case of mixed human/computer systems or self-interested collections of agents).[7] Work on tightly coupled distributed systems (i.e., shared memory multiprocessors) is also outside the scope of this dissertation, although we have had some success using early versions of the work described in this dissertation to schedule blackboard system knowledge source instantiations on true parallel shared memory processors [Decker *et al.*, 1991, Decker *et al.*, 1993a].

There has been work in distributed computing on the scheduling of jobs in loosely coupled distributed systems where a priori fixed decomposition is impossible, job arrivals are stochastic, information about the jobs is uncertain, there is no central authority, etc. The problem is computationally difficult enough to require heuristic solutions (optimal solutions, even when optimality criteria can be defined, are not feasible (see the previous section)). Much of this work has similarities to the distributed scheduling approach of GPGP plus a local scheduler presented in Chapters 5 and 6. However, most decentralized job scheduling approaches make different assumptions than GPGP and operate at a lower level of detail.

For example, Stankovic [Stankovic, 1984b] describes three adaptive decentralized job scheduling algorithms. If we equate a TÆMS task group with a job, these algorithms differ from GPGP family algorithms in several ways. The algorithms are constrained to operate without any a priori knowledge about the jobs (no knowledge about duration or resource usage). Once a job is activated, a separate process scheduler handles the job at a processing node, and the

[7] We do discuss one static task decomposition in Chapter 4.

job scheduler is no longer involved. The jobs are viewed by the scheduler as independent. Each of the algorithms is very simple an efficient, which is important since they are to be used at the operating system level. Similarly to GPGP-style algorithms, jobs may arrive at different rates at different nodes, and the nodes communicate observations about how busy they are (observations that are out of date when they are received). Communication between nodes takes non-negligible amounts of time. Stankovic [Stankovic, 1985] also has discussed decentralized job scheduling using a Bayesian decision theoretic approach.

Work by Cheng, et al. [Cheng *et al.*, 1986] on dynamic, heuristic real-time scheduling has more similarities. Like the GPGP+DTT approach, Cheng's approach deals with groups of tasks (really executable methods in TÆMS terminology) that are interrelated. In Cheng's case these are precedence relationships only. All tasks in a group share a deadline. Tasks in a single group can be divided into clusters that can in turn be executed on different processors. Knowledge of a task group is not available until the task group arrives at some node in the system. Differences between this work and the GPGP+DTT approach include that all (or none) of the tasks in a group must be scheduled, that tasks can execute at any node (homogeneity), and that bidding is used to decide where to transfer task group subclusters.

2.4.5.4 Game Theory

Game theory [Luce and Raiffa, 1958, Rasmusen, 1989] has been used to model decision-making situations faced by interacting agents and to show the existence of certain rational strategies for decision making. The model has been used in distributed AI to show how systems of rational, self-interested agents can, in fact, "cooperate" without such cooperation being designed into them beforehand (the "benevolent agent assumption") [Rosenschein and Genesereth, 1984, Rosenschein and Genesereth, 1985, Genesereth *et al.*, 1986, Rosenschein and Breese, 1989].

This early work represented a decision-making situation as a payoff matrix [Rapoport and Guyer, 1966] showing the utility to be gained by each agent for each possible choice of action. Such a simple representation allows strong results (including careful definitions of what the term "rational" might mean with respect to computational agents [Rosenschein and Genesereth, 1985]), but only under fairly restricted conditions. First, the payoff matrix represents only a single decision. Secondly, most aspects of the problem, such as what choices are possible and the utility of all of the outcomes, are known a priori by all the agents involved.

We can of course represent such simple situations in TÆMS. For example, Figure 2.1 shows a representation of the traditional Prisoner's Dilemma problem. Two agents suspected of a crime are held in separate cells by the authorities. If both agents "cooperate" with one another (keep silent), the authorities can only charge them with a minor crime. If both "defect" (confess), the authorities charge them both. If one cooperates and the other defects, the defector is set free and the other charged. We represent the agents' individual utilities by /tems/ task qualities. The initial qualities ($\mathbf{q}_0$) may change due to interactions between choices as shown in the figure. In general, given the restricted conditions above (that there is only one decision to be made, and that all agents know everything about the decision at hand), it will always be possible to create a payoff matrix from a TÆMS task structure.[8] However, this dissertation will

[8] Thus some styles of deal-making and negotiation, represented in the game theory research, could be easily added to the GPGP set of coordination mechanisms.

focus on environments that consist of schedulable sequences of actions and uncertainty about the local and non-local effects of those actions.

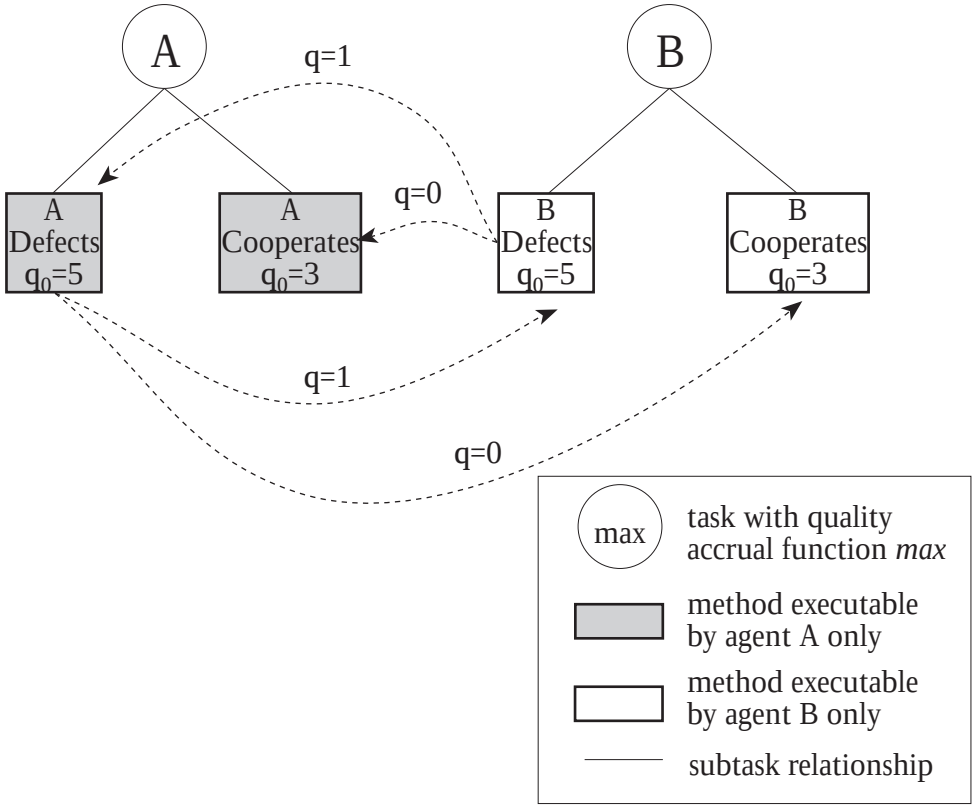

Figure 2.1. Secret agents A and B are captured and must decide whether to "Cooperate" with one another, or to "Defect".

Work has been done on relaxing some of these constraints. For example, Axelrod [Axelrod, 1984] talks about the effect of a continuous sequence of games on solutions to the Prisoner's Dilemma problem (i.e., the evolution of a "tit-for-tat" strategy). Kadane and Larkey [Kadane and Larkey, 1982] discuss the addition of subjective probability. Rosenschein's early and highly influential work has led to more recent results in distributed AI. Rosenschein and Breese have experimented with probabilistic representations of other agents' decisions [Rosenschein and Breese, 1989]. Gmytrasiewicz [Gmytrasiewicz *et al.*, 1991, Durfee *et al.*, 1993] has extended the model to recursively model the beliefs of other agents (the payoff matrix that A believes, the matrix that A believes that B believes, the matrix that A believes that B believes that A believes, etc.)

Still, the basic form of the model which represents a single decision can only easily represent a certain class of problems. Rosenschein and his students have begun working in more complex and general domains. Rosenschein and Zlotkin have coined three terms for these increasingly complex domains: task oriented domains, state-oriented domains, and worth oriented domains. Task oriented domains are those where there is a bundle of non-conflicting

jobs to be done by a set of homogeneous agents (e.g. postal delivery) [Zlotkin and Rosenschein, 1990, Zlotkin and Rosenschein, 1991]. State oriented domains have specific goals that the agents wish to achieve, and actions may interfere with one another (e.g. blocks world planning) [Ephrati and Rosenschein, 1994]. Worth oriented domains have some utility function for world states, so that trade-offs are possible [Goldman and Rosenschein, 1993]. While we have not yet worked on multi-agent planning and negotiation, TÆMS can obviously represent worth-oriented environments and by inclusion the simpler types as well.

2.5 DAI Modeling

Up until very recently, there have been absolutely no analyses of distributed AI systems that have the generality of those in this dissertation, which were produced by our methodology. The sole exception is Sandip Sen's work on distributed meeting scheduling [Sen and Durfee, 1994]. This work was done at about the same time as my work. Sen analyzed the performance (success ratio in scheduling n meetings) and efficiency (cost of communication and total time to schedule) of several meeting scheduling heuristics. These heuristics included different announcement strategies, bidding strategies, and meeting confirmation strategies. Sen also uses a probabilistic view of the environment, in this case the current schedules of two agents for a fixed time period, to compute formulas for the success ratio and total time. Sen verifies these using a simulation of two agents. Unlike our work, Sen actually runs the two agents exhaustively on every possible starting condition ('episode', in our terminology).

2.6 Summary

The form of the TÆMS framework that I am about to describe in this dissertation is more detailed in structure than many organizational-theoretic models of organizational environments, such as Thompson's notions of pooled, sequential, and reciprocal processes [Thompson, 1967], Burton and Obel's linear programs [Burton and Obel, 1984], or Malone's queuing models [Malone, 1987], but is influenced by them, and by the importance of environmental uncertainty, variance, and dependency that appear in contingency-theoretic and open systems views of organizations [Lawrence and Lorsch, 1967, Galbraith, 1977, Stinchcombe, 1990, Scott, 1987].

I started this chapter telling you that building task environment models would allow us to say much more precise things about systems, and might allow us to design coordination algorithms that are appropriate for the task environment at hand. I also told you that the form of that task environment model was inspired by social conceptions of task environments as well as DAI research. The bulk of the chapter then discussed these social conceptions: first I discussed *coordination behavior* as conceived by four different schools of thought, and secondly I discussed *how the environment has an impact on coordination behavior*, again from several points of view. The chapter concluded with a discussion of the DAI view of coordination behaviors and several example systems.

In the next chapter I will introduce and describe in detail TÆMS, which is my modeling framework for representing and reasoning about task environments. The following chapters will show TÆMS being used for mathematical analysis in a simplified distributed sensor network, and for simulation in an implementation of Generalized Partial Global Planning.

CHAPTER 3

A FRAMEWORK FOR MODELING TASK ENVIRONMENTS

> *[and his students] . . . created a world of the mind, of the intimate imagination, which is as real in its way as any actual country on the map. Sir Karl Popper, in one of his most important papers, calls it 'the third world,' or 'world three.' The first world is the objective world of things. The second world is my inner, subjective world. But, says Popper, there's a third world, the world of objective contents of thoughts. Teilhard de Chardin calls this third world the* noosphere*, that is, the world of the mind.*
>
> *— Sampled from an unknown source by The Orb, u.f.orb:"o.o.b.e", 1992*

This chapter will discuss the characterization of the features, and model of the processes, in computationally intensive task environments called TÆMS (Task Analysis, Environment Modeling, and Simulation) [Decker and Lesser, 1993d, Decker and Lesser, 1993e, Decker, 1994c]. No characterization currently exists that formally captures the range of features, processes, and especially interrelationships that occur in computationally intensive AI task environments. In the simplest terms, a TÆMS model of a task environment specifies what actions are available to agents and how those actions relate to one another and to the performance of the system as a whole. The TÆMS framework is useful not only for the study of coordination and other related CDPS behaviors (e.g., negotiation), but also for the study of the planning and scheduling of computation in realistic real-time or parallel task environments [Garvey *et al.*, 1993, Garvey and Lesser, 1993].

The reason we have created the TÆMS framework is rooted in the desire to produce general theories in AI [Cohen, 1991]. Consider the difficulties facing an experimenter asking under what environmental conditions a particular local scheduler produces acceptable results, or when the overhead associated with a certain coordination algorithm is acceptable given the frequency of particular subtask interrelationships. At the very least, our framework provides a characterization of environmental features and a concrete, meaningful language with which to state correlations, causal explanations, and other forms of theories. The clear specification of a computational task environment also allows the use of very strong analytic or experimental methodologies, including paired-response studies, ablation experiments, and parameter optimization. TÆMS exists as both a language for stating general hypotheses or theories and as a system for simulation. The simulator is written in standard, object-oriented Common Lisp (CLOS) for any platform. The simulator also supports the graphical display of generated task structures, agent actions, and statistical data collection on the TI Explorer and DEC Alpha, via CLIM (Common Lisp Interface Manager) and CLIP (Common Lisp Instrumentation Package [Westbrook *et al.*, 1994]).

This chapter will describe TÆMS in detail. First I will give a brief overview, then introduce the core modeling framework. Next, I will introduce the DSN (Distributed Sensor Network)

environment that I will use for the examples in the later chapters (and which I will analyze in Chapter 4). Section 3.4 will then describe the language for building objective models of task structure instances in an environment. Section 3.5 will describe the way that TÆMS models the subjective views that agents have about the objective task structure instances in the previous chapter. Finally, Section 3.6 will briefly describe the ways that TÆMS could model the processes of statistically generating objective and subjective task structures in an environment. That section will also give examples of models outside of the DSN domain, such as hospital patient scheduling.

3.1 General Framework

The principle purpose of a TÆMS model is to analyze, explain, or predict the performance of a system or some component. While TÆMS does not establish particular performance criteria, it focuses on providing two kinds of performance information: the temporal intervals of task executions, and the *quality* of the execution or its result. *Quality* is an intentionally vaguely-defined term that must be instantiated for a particular environment and set of performance criteria. Examples of *quality* measures include the precision, belief, or completeness of a task result. We will assume that *quality* is a single numeric term with an absolute scale, although the algebra can be extended to vector terms. In a computationally intensive AI system, several quantities—the quality produced by executing a task, the time taken to perform that task, the time when a task can be started, its deadline, and whether the task is necessary at all—are arbitrarily affected by the execution of other tasks. In real-time problem solving, alternate task execution methods may be available that trade-off time for quality. Agents do not have unlimited access to the environment; what an agent believes and what is really there may be different.

From the point of view of TÆMS an agent is a locus of belief and action. An agent may "know" or "believe" in the existence of a certain task structure; the agent has a state. A computational agent might have a "belief database" where it stores information about the task structure.[1] An agent may also execute methods that are part of this task structure it believes —this is called a "method execution action". For example, Agent 1 might gain knowledge of the task structure in a certain problem solving episode. That task structure might contain an executable method $M1$. If Agent 1 executes this method, it will take some amount of time (as specified in the model) and the agent will accrue some amount of quality (as specified in the model). The action may have non-local effects on the actions (past, current, and future) of other agents in the environment.

The model of environmental and task characteristics proposed has three levels: *generative*, *objective*, and *subjective*. The *generative* level describes the statistical characteristics required to generate the objective and subjective episodes in an environment; it is a workload generator. A generative level model consists of a description of the generative processes or distributions from which the range of alternative problem instances can be derived, and is used to study

[1] For the rest of this dissertation, we will usually refer to information the agent has as "belief" and not "knowledge", since agents will often have information in their databases that is false (but believed to be true). This is a very common epistemological distinction between knowledge and belief [Halpern and Moses, 1985, Halpern, 1986]. Since in almost any modal logic of knowledge and belief $\mathit{Knows}(x) \rightarrow \mathit{Believes}(x)$, we don't lose any generality.

performance over a range of problems in an environment, avoiding single-instance examples. The *objective* level describes the essential, 'real' task structure of an episode (a particular problem-solving situation or instance) over time. It is roughly equivalent to a formal description of a single problem-solving situation such as those presented in [Durfee and Lesser, 1991], without the information about particular agents. The *subjective* level describes the agent's local view of the objective problem-solving situation over time (e.g., how much does an agent know about what is really going on, and how much does it cost to find out—where the uncertainties are from the agent's point of view). The subjective level is essential for evaluating control algorithms, because agents must make decisions with only subjective information about the current episode.[2] In the rest of this chapter, we first describe the *objective* level, because the other two are linked to its definitions (i.e., generating objective task structures, and determining an agent's subjective view of an objective task structure). The objective level is the core of TÆMS.

3.2 TÆMS model summary

This section will lay out the mathematical details of the model without any examples. Readers who would prefer to be introduced to TÆMS more slowly, through the use of a DSN example, should skip ahead to the next section. All of the information summarized in this section is repeated later in the chapter. The following description is top-down, starting with the environment and ending with what the agent perceives. Thus we do not define a coordination problem until the final subsection. TÆMS models are discrete state-based models, where the state of all agents in the system at time $t + 1$ is computed from their state at all previous times.

An *environment* $\mathcal{E}$ is a generator function that takes and arbitrary set of parameters (including the set of agents), and produces *episodes* $\mathbf{E}$. This is done according to the generative model for the specific environment; several examples, including a random environment generator, are given later in this chapter. A continuous environment produces episodes with some frequency (a parameter), a discrete environment produces a new episode upon each invocation.

3.2.1 Objective task structure summary

We will now describe the objective task structure of a problem-solving episode. An *episode* $\mathbf{E}$ consists of a set of *task groups* $\mathcal{T}$; $\mathbf{E} = \langle \mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_n \rangle$. Each task group has an arrival time $\mathrm{Ar}(\mathcal{T})$, and a deadline $\mathrm{D}(\mathcal{T})$. A task group represents a set of computationally related actions.[3] A *task group* is represented by a directed acyclic graph. The nodes of the graph are called *tasks* T. One task is denoted the root task, and is usually simply indicated by the symbol for the entire task group, $\mathcal{T}$. Tasks that have no children are called *executable methods*, or just *methods* M for short. Tasks that do have children, but that are not the root task, are straightforwardly called *subtasks*. The structure of a task group is meant to reflect the problem's task structure.

The edges of this graph form the `subtask` relationship. Task or task group quality at a given time ($Q(T, t)$) is based on the *subtask* relationship. This quality function is constructed

[2] In organizational theoretic terms, subjective *perception* can be used to predict agent actions or *outputs*, but unperceived, objective environmental characteristics can still affect performance (or *outcomes*) [Scott, 1987].

[3] For example, if the environment is the USTravel business in the UMass Campus Center, an episode might consist of two task groups: one to handle the Westbrook's trip to Disney World, and another task group to handle everything related to Garvey's trip to Seattle.

recursively. Formally, the subtask relationship is defined as $\mathsf{subtask}(T, \mathbf{T}, Q)$, where $\mathbf{T}$ is the set of all direct subtasks of T and Q is a quality function $Q(T, t) : [\text{tasks} \times \text{times}] \mapsto [\text{quality}]$ that returns the quality associated with T at time t. The semantics of a particular environment are modeled by the appropriate choice of the quality function Q (e.g., minimum, maximum, summation, or the arithmetic mean). In particular, we will often write of the quality achieved at a task group at time t, $Q(\mathcal{T}, t)$, meaning the current quality at the root task.

Executable methods represent domain actions, like executing a blackboard knowledge source, running an instantiated plan, or executing a piece of code with its data.[4] Executable methods have several functions defined on them. $\mathbf{q}(M, t)$ is the *current maximum quality* that can be achieved by executing M at time t for its duration $\mathbf{d}(M, t)$. $\mathbf{d}(M, t)$ is the *current duration* of method M at time t. $\mathrm{Progress}(M, t)$ is the number of time units spent executing M. The definitions of $\mathbf{q}(M, t)$, $\mathbf{d}(M, t)$, and $\mathrm{Progress}(M, t)$ are fixed by TÆMS. $Q(M, t)$ is the quality at M at time t. This function is available for modeling a particular environment, but it is constrained to obey the identity:

$$Q(M, t) = \mathbf{q}(M, t) \text{ if } [\mathrm{Progress}(M, t) = \mathbf{d}(M, t)] \wedge [\mathrm{Finish}(M) \leq \mathrm{D}(M)]$$

(i.e. the quality is the maximum quality if the executable method was completed before its deadline).

Any task T containing a method that starts executing before the execution of another method M finishes may potentially affect M's execution through a *non-local effect* e. We write this relation (a labeled arc in the task structure graph) as $\mathsf{nle}(T, M, e, p_1, p_2, \ldots)$, where the p's are parameters specific to a class of effects. There are precisely two possible outcomes of the application of a non-local effect on M under our model: *duration effects* where $\mathbf{d}(M, t)$ (duration) is changed and *quality effects* where $\mathbf{q}(M, t)$ (maximum quality) is changed. An effect class e is thus a function $e(T, M, t, d, q, p_1, p_2, \ldots) : [\text{task} \times \text{method} \times \text{time} \times \text{duration} \times \text{quality} \times \text{parameter 1} \times \text{parameter 2} \times \ldots] \mapsto [\text{duration} \times \text{quality}]$.

Computing $\mathbf{d}(M, t)$ *and* $\mathbf{q}(M, t)$. Each method has an initial maximum quality $\mathbf{q}_0(M)$ and duration $\mathbf{d}_0(M)$ so we define $\mathbf{q}(M, 0) = \mathbf{q}_0(M)$ and $\mathbf{d}(M, 0) = \mathbf{d}_0(M)$. If there is only one non-local effect with M as a consequent $\mathsf{nle}(T, M, e, p_1, p_2, \ldots)$, then $[\mathbf{d}(M, t), \mathbf{q}(M, t)] \leftarrow e(T, M, t, \mathbf{d}(M, t-1), \mathbf{q}(M, t-1), p_1, p_2, \ldots)$. If there is more than one non-local effect, then the effects are applied one after the other in an order specified in the model. We will define sixteen example non-local effects in this chapter.

3.2.2 The subjective mapping and agent actions

We define a *subjective mapping* $\varphi : [(x \in \mathbf{E}) \times A \times t] \mapsto x'$ that maps from elements in the current episode to agent A's subjective view of those elements. The mapping may be empty for an element. This is the core of the subjective level model. Later in the chapter we discuss a simple default definition for φ.[5]

Agents can perform three types of actions: they can execute methods, they can communicate with one another, and they can perform "information gathering actions". All of these actions take some amount of time as specified in the model. For example, executing the method

[4] An executable method is always an action with context—thus "X-ray Nagi's foot" is an executable method, while "X-ray" is not.

[5] One obvious extension is to make φ return a set of elements, not just one.

M at time t takes $\mathbf{d}(M, t)$, as we defined earlier. The main or "local" effect of this execution is the production of some amount of quality at that node in the task structure ($Q(M, t)$). This may also change the quality at the task group root ($Q(\mathcal{T}, t)$). Executing a method may also have secondary, or non-local effects, as defined earlier. A method might enable, hinder, facilitate, etc., other methods in the task structure by changing their current duration and maximum quality. Method execution actions are always part of the environment being modeled.

Communication actions and information gathering actions can be part of the environment or can be meta-actions. Agents can communicate the results of any task. We define a special quality function, $Q_{\mathbf{avail}}(T, A, t)$ that returns the maximum of the quality available locally ($Q(T, t)$) and the quality that has been communicated to the agent. Communication takes some fixed amount of time between agents as specified in the model.[6]

Let Γ_A^t denote agent A's set of beliefs at time t (the agent's belief database). External information gets into the agent's belief database via an information gathering action. The default information gathering action places all communications that have arrived in the belief database, and places the subjective version ($\varphi(x, A, t)$) of any newly-arrived elements of the current episode in the belief database.

3.2.3 Coordination Problems

We can define a performance measure $\mathcal{P}(\mathbf{E})$ for the system (or for an agent) that is a function of the episode. The default is the sum of the task group qualities ($\mathcal{P}(\mathbf{E}) = \sum_{\mathcal{T} \in \mathbf{E}} Q(\mathcal{T}, \mathrm{D}(\mathcal{T}))$). We can also define a control function (alternately, a "strategy", decision rule, or control policy) for each agent that given the agent's current beliefs and the time will return the action that the agent should take at that time. One statement of a cooperative coordinated control problem (similar in spirit to the specification of a team theory decision problem) is then to find a set of such control functions, one for each agent in an environment, so as to attempt to maximize the expected performance measure for the entire system of agents. A similar problem studied in DAI is the multi-agent self-interested coordination problem, where we choose an agent's control function so as to attempt to maximize only it's own performance (with the realization that other agents are doing likewise).

To summarize, using the TÆMS framework involves building a *generative* model of an environment that produces episodes that in turn comprise *objective* task structures. A *subjective* model describes a mapping from these objective structures to the information available to the agents (if the agents execute an information gathering action). All of this together is the description of the environment that can be used to design and analyze coordination mechanisms. The next section will introduce our in-depth example environment, a distributed sensor network environment. Chapter 4 will analyze this environment and two control functions in detail. The rest of this chapter will focus first on the specification of objective task structures—the most complex and fully developed part of TÆMS. Then this chapter will discuss subjective and generative models, and give several short examples from many different problem solving environments.

[6] Obvious extensions include extra complexity in the communication subnet model: lost messages, random variable delays, connectivity delays, etc.

3.3 Distributed Sensor Network Example

Before we discuss formally the basic components of an objective model, we turn to an example environment for which we will build a model using the TÆMS framework. This example grows out of the set of single instance examples of distributed sensor network (DSN) problems presented in [Durfee *et al.*, 1987]. The authors of that paper compared the performance of several different coordination algorithms on these examples, and concluded that no one algorithm was always the best—not a very surprising result and one that can be viewed as the central tenet of contingency theory. This is the classic type of result that the TÆMS framework was created to address—we will *explain* this result, and better yet, *predict* which algorithm will do the best in each situation. In Chapter 4 we do this by extending the analysis to take into account a statistical view of the environmental characteristics. The level of detail to which you build a model in TÆMS will depend on the question you wish to answer—here we wish to identify the characteristics of the DSN environment, or the organization of the agents, that cause one algorithm to outperform another.

In a DSN problem (as you may recall from Chapter 1), the movements of several independent vehicles will be detected over a period of time by one or more distinct sensors, where each sensor is associated with an agent. The performance of agents in such an environment is based on how long it takes them to create complete vehicle tracks, including the cost of communication. The organizational structure of the agents will imply the portions of each vehicle track that are sensed by each agent.

Our task environment model of naturally distributed problems assumes that several independent groups of tasks arrive at multiple locations over a period of time (the episode). For example, in a distributed sensor network (DSN) episode the movements of several independent vehicles will be detected over a period of time by one or more distinct sensors, where each sensor is associated with an agent. The performance of agents in such an environment will be based on how long it takes them to process all the data into complete vehicle tracks, including the cost of communicating sensor data, intermediate task results, and meta-level communication, if any. The organizational structure of the agents will imply which subsets of which task groups are available to which agents and at what cost. For example, if DSN agents have overlapping sensors, multiple agents can potentially work on data in the overlapping area (from its own sensor) without any extra communication costs. We make several simplifying assumptions in this DSN model: that the agents are homogeneous (have the same capabilities with respect to receiving data, communicating, and processing tasks), that the agents are cooperative (interested in maximizing the system performance over maximizing their individual performance), that the data for each episode is available simultaneously to all agents as specified by their initial organization, and that there are only structural (precedence) constraints within the subtasks of each task group.[7]

Any single episode can be specified by listing the task groups, and what part of each task group was available to which agents, given the organizational structure. Our analyses in Chapter 4 will be based on the statistical properties of episodes in an environment, not any single instance of an episode. The properties of the episodes in a DSN environment are summarized by the tuple $\mathcal{D} =< A, \eta, r, o, \mathcal{T}(l) >$ where A specifies the variable number of

[7] In general there are usually more complex interrelationships between subtasks that affect scheduling decisions, such as *facilitation*.

agents, η the expected number of task groups, r and o specify the variable structural portion of the organization by the *range* of each agent and the *overlap* between agents, and $\mathcal{T}(l)$ specifies a template for the structure of each task group. Our DSN model will represent each vehicle track as a separate task group, thus $\mathcal{T}(l)$ corresponds to a vehicle track of length l. DSN agent sensors are assumed to be square, and are arranged as a square. The range parameter specifies the physical range of the agent's sensor (the width of a sensed square). The overlap parameter specifies how wide the overlapping area sensed by two nearby agents is ($0 \leq o \leq r$). Figure 3.1 shows an example. Later sections will describe in detail how the task group structures are specified.

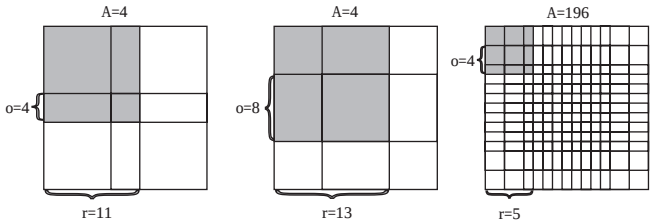

Figure 3.1. Examples of 18×18 DSN organizations

If the summary of a DSN environment is the tuple $\mathcal{D} =< A, \eta, r, o, \mathcal{T}(l) >$, then a particular episode in this environment can be described by the tuple $\mathbf{D} =< \mathcal{T}_1(l_1), \ldots, \mathcal{T}_n(l_n) >$, where n is a random variable generated from a Poisson distribution with location parameter (central tendency) of η. The l_i are generated by a more complex process. For each track/task group, two random points on two different sides of the bounding box around the agents' sensors are picked and a line is computed between them. Chapter 4 will analyze this distribution in some detail. We have just described how to generate an episode $\mathbf{D}$ in any DSN environment $\mathcal{D}$, by generating values for n and $l_1 \ldots l_n$. This is the generative model.

In our objective-level model of DSN problem instances, each vehicle track is modeled as a task group T_i. Several task groups (vehicle tracks) may occur simultaneously in a single problem solving episode. The simplest objective model is that each task group $\mathcal{T}_i$ is associated with a track of length l_i and has the following objective structure, based on a simplified version of the DVMT:

- (l_i) vehicle location methods (VLM) that represent processing raw signal data at a single location resulting in a single vehicle location hypothesis
- ($l_i - 1$) vehicle tracking methods (VTM) that represent short tracks connecting the results of the VLM at time t with the results of the VLM at time $t + 1$
- One vehicle track completion method (VCM) that represents merging all the VTMs together into a complete vehicle track hypothesis

Non-local effects, which relate the executions of tasks to one another, exist as shown in Figure 3.2—two VLMs enable each VTM, and all VTMs enable the lone VCM. This

structure is fairly common in many other environments, where a large amount of detailed work needs to be done, the results of which are collected at a single location or agent and processed again (integrated), and so on up a hierarchy. Coordination is used not only to accomplish the necessary transfer of results from one agent to another, but to avoid redundant work on the part of agents with overlapping capabilities, and also to potentially balance the workloads.

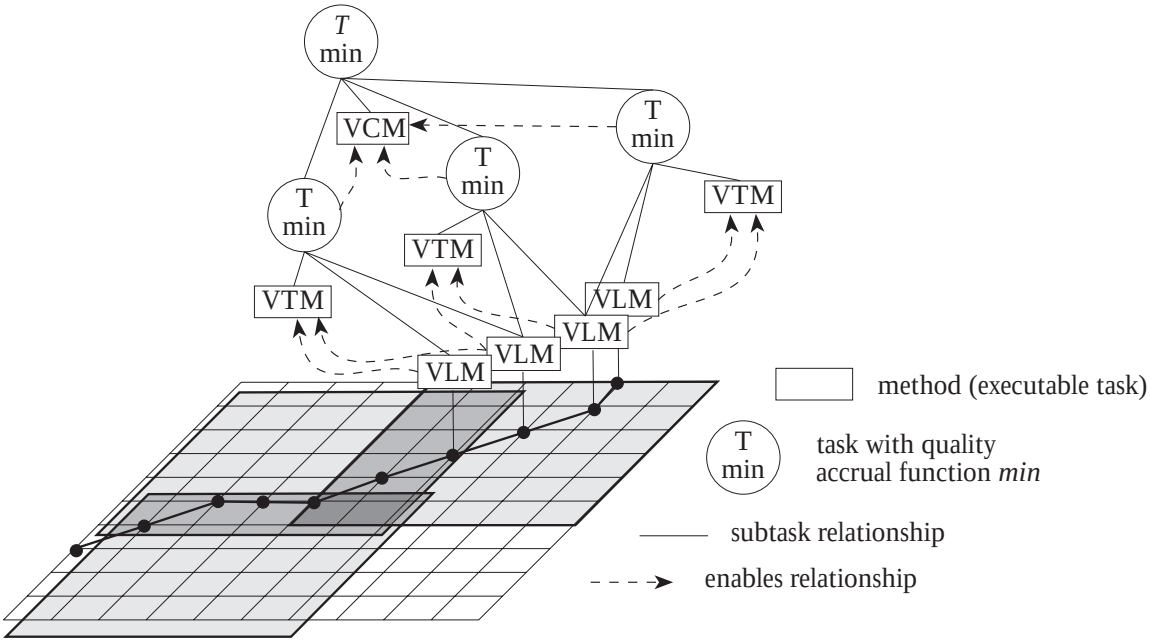

Figure 3.2. Objective task structure associated with a single vehicle track.

We have used this model to develop expressions for the expected value and distribution of the time of termination of a set of agents in any arbitrary DSN environment that has a static organizational structure and coordination algorithm (Chapter 4). We have also used this model to analyze a dynamic, one-shot reorganization algorithm (and have shown when the extra overhead is worthwhile versus the static algorithm. In each case we can predict the effects of adding more agents, changing the relative cost of communication and computation, and changing how the agents are organized (in the range and overlap of their capabilities). These results were achieved by direct mathematical analysis of the model and verified through simulation in TÆMS. We will describe these results later (Chapter 4), after discussing more details about the the objective, subjective, and generative levels in this chapter.

3.4 TÆMS Objective Level Models

The *objective* level describes the essential structure of a particular problem-solving situation or instance over time. It focuses on how task interrelationships dynamically affect the *quality* and *duration* of each task. The basic idea is that *task groups* (problem instances) appear in an episode at some frequency, and induce tasks T to be executed by the agents under study.

Task groups are computationally independent of one another, but tasks within a single task group have interrelationships. Task groups and tasks may have deadlines $D(T)$. The *quality* of the execution or result of each task influences the *quality* of the task group result $Q(T, t)$ in a formally-defined way (Section 3.4.1). These quantities can be used to evaluate the performance of a system.

An individual task that has no subtasks is called a method M and is the smallest schedulable chunk of work (though some scheduling algorithms will allow some methods to be preempted, and some schedulers will schedule at multiple levels of abstraction). There may be more than one method to accomplish a task, and each method will take some amount of time and produce a result of some *quality*. Quality of an agent's performance on an individual task is a function of the timing and choice of agent actions ('local effects'), and possibly other (previous or future) task executions ('non-local effects'). When local or non-local effects exist between tasks that are known by more than one agent, we call them *coordination relationships*. The basic purpose of the objective model is to formally specify how the execution and timing of tasks affect the measure of quality.

3.4.1 Local Effects: The Subtask Relationship

Task or task group quality ($Q(T, t)$) is based on the *subtask* relationship. This quality function is constructed recursively—each task group consists of tasks, each of which consists of subtasks, etc.—until individual executable tasks (methods) are reached. Formally, the subtask relationship is defined as $\mathsf{subtask}(T, \mathbf{T}, Q)$, where $\mathbf{T}$ is the set of all direct subtasks of T and Q is a quality function $Q(T, t) : [\text{tasks} \times \text{times}] \mapsto [\text{quality}]$ that returns the quality associated with T at time t. In a valid model, the directed graph induced by this relationship is acyclic (no task has itself for a direct or indirect subtask). A task with no subtasks is called a *method*. Methods represent computation or physical actions; quality accrual at methods is described in the next section.

The semantics of a particular environment are modeled by the appropriate choice of the quality function Q (e.g., minimum, maximum, summation, or the arithmetic mean). For example, if $\mathsf{subtask}(T_1, \mathbf{T}, Q_{\min})$, then $Q(T_1, t) = Q_{\min}(T_1, t) = \min_{T \in \mathbf{T}} Q(T, t)$. In this case the quality that is associated with task T_1 is the minimum quality associated with any of its subtasks. The four most popular non-method task quality combination functions are the following:

Minimum. The quality that is generated is the minimum subtask value. This is sometimes referred to as an AND (but see also *Average* below) because the quality of the parent remains at a minimum until every subtask has been completed. Often, this is used when all subtasks of the task must be completed before any value is generated (by making each method have quality 0 before it is executed).

$$Q_{\min}(T_1, t) = \min_{T \in \mathbf{T}} Q(T, t) \tag{3.1}$$

Maximum. This is sometimes referred to as an XOR or OR (but see also *Sum* below) because only one subtask needs to be executed to gain quality at the parent task. Often this is used to model the situation where only one of the subtasks must be completed to receive value. One can construct situations where completing additional subtasks may generate additional value. For example, if two tasks have potential maximum qualities of 10 and 5, and the agent

achieves the 5, the agent may at a later time come back and achieve the 10 (adding 5 to the quality of the parent).

$$Q_{\max}(T_1, t) = \max_{T \in \mathbf{T}} Q(T, t) \tag{3.2}$$

Sum. This means that the quality generated for a task is the sum of the qualities generated by its subtasks. Another common meaning for OR.

$$Q_{\Sigma}(T_1, t) = \sum_{T \in \mathbf{T}} Q(T, t) \tag{3.3}$$

Average. This means that the quality generated for a task is the mean of the quality generated by its subtasks. This is what the DVMT considered an AND.

$$Q_{\mathrm{avg}}(T_1, t) = \frac{\sum_{T \in \mathbf{T}} Q(T, t)}{|\mathbf{T}|} \tag{3.4}$$

Other functions may be used for modeling particular environments.[8] Another possible function is one that represents diminishing returns to a fixed maximum, similar to the linear Q_{lin} function defined in the next section, or various convex curves such as $\frac{1}{1-e^t}$. Functions like sum and average indicate the possibility that not all tasks in the environment need to be carried out. Computational task structures may represent search processes where agents are satisficing, not optimizing. We have now described how quality is modeled at tasks that have subtasks, and now turn our attention to methods.

3.4.2 Local Effects: Method Quality

Each method M at a time t will potentially produce (if executed by an agent, see Section 3.5.3.1) some *maximum quality* $\mathbf{q}(M, t)$ after some amount of elapsed time $\mathbf{d}(M, t)$ (we will defer any further definition of the functions $\mathbf{d}$ and $\mathbf{q}$ until we discuss non-local effects in Section 3.4.3). The execution of methods is interruptible, and if multiple methods for a single task are available, the agent may switch between them (typically, alternative methods tradeoff time and quality). We model the effect of interruptions, if any, and the reuse of partial results as non-local effects (see Section 3.4.3).

[8] The set of possible aggregation operators include three basic classes: *conjunctions*, *disjunctions*, and *trade-offs*. Dubois and Prade have shown that the Triangular norms (including $\min$), averaging operators (including mean), and Triangular conorms (including $\max$ and summation) are the most general families of binary functions that respectively satisfy the semantic requirements of the three basic classes of aggregation operators [Bonissone and Decker, 1986, Dubois and Prade, 1984].

Let $\mathrm{Progress}(M,t)$ be the current amount of progress on the execution of M. If M were not interruptible and $\mathrm{Start}(M)$ and $\mathrm{Finish}(M)$ were the execution start time and finish time, respectively, of M, then:

$$\mathrm{Progress}(M,t) = \begin{cases} 0 & t \leq \mathrm{Start}(M) \\ t - \mathrm{Start}(M) & \mathrm{Start}(M) < t < \mathrm{Finish}(M) \\ \mathrm{Finish}(M) - \mathrm{Start}(M) & t \geq \mathrm{Finish}(M) \end{cases}$$

We typically model the quality produced by a method $Q(M,t)$ using a linear growth function Q_{lin}:

$$Q_{\mathrm{lin}}(M,t) = \begin{cases} \frac{\mathrm{Progress}(M,t)}{\mathbf{d}(M,t)}(\mathbf{q}(M,t)) & \mathrm{Progress}(M,t) < \mathbf{d}(M,t) \\ \mathbf{q}(M,t) & \mathrm{Progress}(M,t) \geq \mathbf{d}(M,t) \end{cases}$$

Other models (besides linear quality functions) have been proposed and are used, such as concave quality functions (must execute most of a task before quality begins to accumulate), convex quality functions (most quality is achieved early on in a method, and only small increases occur later), and 'mandatory and optional parts' quality functions [Liu *et al.*, 1991]. The desired $Q(M,t)$ can be easily defined for any of these.

As an example of the power of this representation, we consider the two main schools of thought on quality accumulation: the anytime algorithm camp [Boddy and Dean, 1989] and the design-to-time (approximate processing) camp[Decker *et al.*, 1990, Garvey and Lesser, 1993]. We can represent their ideas succinctly; in the anytime algorithm model partial results are always available,[9] as in the definition of $Q_{\mathrm{lin}}(M,t)$ above, while in the design-to-time model results are not available (quality does not accrue) until the task is complete, as in the definition of $Q_{\mathrm{DTT}}(M,t)$:

$$Q_{\mathrm{DTT}}(M,t) = \begin{cases} 0 & \mathrm{Progress}(M,t) < \mathbf{d}(M,t) \\ \mathbf{q}(M,t) & \mathrm{Progress}(M,t) \geq \mathbf{d}(M,t) \end{cases}$$

Another difference between design-to-time (DTT) and other approaches will show up in our generative and subjective additions to this model—DTT does not assume that $Q(M,t)$ is fixed and known, but rather that it is an estimator for the actual method response. Finally, deadlines can be associated with task groups or individual tasks, and quality accumulation defined such that any work done on a task after its deadline does not increase quality.

3.4.3 Non-local Effects

Any task T containing a method that starts executing before the execution of another method M finishes may potentially affect M's execution through a *non-local effect* e. We write this relation $\mathsf{nle}(T, M, e, p_1, p_2, \ldots)$, where the p's are parameters specific to a class of effects. There are precisely two possible outcomes of the application of a non-local effect on M under our model:

duration effects: where $\mathbf{d}(M,t)$ (duration) is changed

quality effects: where $\mathbf{q}(M,t)$ (maximum quality) is changed.

[9] In Boddy's paper, the assumption is made that $Q(M,t)$ has monotonically decreasing gain.

An effect class e is thus a function $e(T, M, t, d, q, p_1, p_2, \ldots)$: [task $\times$ method $\times$ time $\times$ duration $\times$ quality $\times$ parameter 1 $\times$ parameter 2 $\times \ldots] \mapsto$ [duration $\times$ quality].

The amount and direction of an effect is dependent on the relative timing of the method executions, the quality of the effect's antecedent task, and whether information was communicated between the agents executing the methods (in multi-agent models). Some effects are continuous, depending on the current quality of the effect's antecedent $Q(T, t)$. Some effects are triggered by a rising edge of quality past a threshold; for these effects we define the helper function $\Theta(T, \theta)$ that returns the earliest time when the quality surpasses the threshold: $\Theta(T, \theta) = \min(t)$ s.t. $Q(T, t) > \theta$.

Communication. Some effects depend on the availability of information to an agent. We indicate the communication of information at time t about task T_a to an agent A with a delay of δ_t by $\mathsf{comm}(T_a, A, t, \delta_t)$. There are many models of communication channels that we could take for a communication submodel; since it is not our primary concern we use a simple model with one parameter, the time delay δ_t.[10] For defining effects that depend on the availability of information, we define the helper function $Q_{\text{avail}}(T, t, A)$ that represents the quality of a task T 'available' to agent A at time t. If T was executed at A, $Q_{\text{avail}}(T, t, A) = Q(T, t)$. If T was executed (or is being executed) by another agent, then the 'available' quality is calculated from the last communication about T received at agent A prior to time t.

Computing $\mathbf{d}(M, t)$ *and* $\mathbf{q}(M, t)$. Each method has an initial maximum quality $\mathbf{q}_0(M)$ and duration $\mathbf{d}_0(M)$ so we define $\mathbf{q}(M, 0) = \mathbf{q}_0(M)$ and $\mathbf{d}(M, 0) = \mathbf{d}_0(M)$. If there is only one non-local effect with M as a consequent $\mathsf{nle}(T, M, e, p_1, p_2, \ldots)$, then $[\mathbf{d}(M, t), \mathbf{q}(M, t)] \leftarrow e(T, M, t, \mathbf{d}(M, t-1), \mathbf{q}(M, t-1), p_1, p_2, \ldots)$. If there is more than one non-local effect, then the effects are applied one after the other in an order specified in the model.

The maximum quality function $\mathbf{q}$ can also be defined for tasks or task groups. The precise definition depends on the set of quality accrual functions in the model. Using the four quality accrual functions we have already discussed (minimum, maximum, summation, mean) the definition of maximum quality for a non-method task $\mathbf{q}(T, t)$ is as follows:

$$\mathbf{q}(T, t) = \begin{cases} \max_{x \in \mathbf{T}} \mathbf{q}(x, t) & \text{if } \mathsf{subtask}(T, \mathbf{T}, Q_{\max}) \\ \min_{x \in \mathbf{T}} \mathbf{q}(x, t) & \text{if } \mathsf{subtask}(T, \mathbf{T}, Q_{\min}) \\ \sum_{x \in \mathbf{T}} \mathbf{q}(x, t) & \text{if } \mathsf{subtask}(T, \mathbf{T}, Q_{\sum}) \\ \frac{\sum_{x \in \mathbf{T}} \mathbf{q}(x, t)}{|\mathbf{T}|} & \text{if } \mathsf{subtask}(T, \mathbf{T}, Q_{\text{mean}}) \end{cases}$$

The current duration function $\mathbf{d}$ has no meaningful objective definition when applied to non-method tasks. "Maximum duration" could be defined, but is generally a useless concept. A more useful concept for scheduling—the minimum duration required for achieving maximum quality at a task—is explored in [Garvey *et al.*, 1993]. The clear specification of such concepts is one of the benefits of using our framework.

3.4.3.1 Non-local Effect Examples

Non-local effects are the most important part of the TÆMS framework, since they supply most of the characteristics that make one task environment unique and different from another.

[10] Other parameters, such as channel reliability, could be added. The description of an agent's control and coordination algorithms will describe when and where communication actually occurs (see communication actions in Section 3.5.3.2, and the concept of agency in Section 3.5.1).

Typically a model will define different classes of effects, such as causes, facilitates, cancels, resource−constrains, inhibits, and enables [Decker and Lesser, 1992]. This section contains definitions for four common classes of effects that have been useful in modeling different environments. When non-local effects occur between methods associated with different agents, we call them *coordination relationships*.

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

$$\mathsf{enables}(T_a, M, t, d, q, \theta) = \begin{cases} [\infty, 0] & t < \Theta(T_a, \theta) \\ [\mathbf{d}_0(M), \mathbf{q}_0(M)] & t \geq \Theta(T_a, \theta) \end{cases} \tag{3.5}$$

Facilitates. Another effect, used by the PGP algorithm [Durfee and Lesser, 1991] but never formally defined, is the facilitates effect. In organization theory, Stinchcombe defines facilitation as an 'external effect' in relation to the spatial properties of environmental activities [Stinchcombe, 1987], but also recognizes computational facilitation through the transmission of information. Computationally, facilitation occurs when information from one task, often in the form of constraints, is provided that either reduces or changes the search space to make some other task easier to solve. A simple to understand example of this relationship in computation is the relationship between sorting and searching. It is faster to retrieve an item from a sorted data structure, but sorting is not *necessary* for retrieval. Hence the sorting task facilitates the retrieval task.

In our framework, one task may provide results to another task that facilitates the second task by decreasing the duration or increasing the quality of its partial result. Therefore the facilitates effect has two constant parameters (called *power* parameters) $0 \leq \phi_d \leq 1$ and $0 \leq \phi_q \leq 1$, that indicate the effect on duration and quality, respectively. The effect varies not only through the power parameters, but also through the quality of the *facilitating* task available when work on the *facilitated* task starts (the ratio R, defined below). Note that before work is started on a method, $\mathrm{Start}(M) = t$ (i.e. formulae are evaluated as if execution were about to start).

$$\begin{aligned} R(T_a, s) &= \frac{\mathrm{Q}_{\mathrm{avail}}(T_a, s)}{\mathbf{q}(T_a, s)} \\ \mathsf{facilitates}(T_a, M, t, d, q, \phi_d, \phi_q) &= [d(1 - \phi_d R(T_a, \mathrm{Start}(M))), \\ &\quad\ q(1 + \phi_q R(T_a, \mathrm{Start}(M)))] \end{aligned} \tag{3.6}$$

So if T_a is completed with maximal quality, and the result is received before M is started, then the duration $\mathbf{d}(M,t)$ will be decreased by a percentage equal to the duration power ϕ_d of the facilitates effect. The second clause of the definition indicates that communication after the start of processing has no effect. In Chapter 6 we will explore the effects on coordination of a facilitates effect alone and with other effects.

Hinders. The hinders effect is the opposite of facilitates, because it increases the duration and decreases the maximum quality of the consequent. It can be used expressively in a model to represent situations that reduce some baseline performance measures (as opposed to facilitates, which represents an increase in the baseline). Such situations occur computationally when

there are multiple methods for a task and one fails to provide as many constraints as the other. hinders can also be used as a high-level model of distraction [Durfee *et al.*, 1987], where inappropriate constraints communicated from one agent to another lead the second agent on a wild goose chase.

Precedence. We define the precedes effect as a combination of enables and hinders. That is, one task must be done before another *and* the first task must be done well or later tasks will suffer for it. This definition comes about because it models the behavior of the DVMT. If T_a precedes M, then M has infinite duration and 0 maximum quality until some quality is accrued at T_a. Afterwards, the duration drops toward the initial value and the maximum quality increases to the initial value according to the ratio of available and maximum quality and the precedence effect's power parameters ($\phi_d \geq 0$ and $\phi_q \geq 0$). The following formula is more easily understood if one keeps in mind that, in general, the ratio of available quality to maximum quality will go from 0 to 1 as methods are executed.

$$\text{precedes}(T_a, M, t, d, q, \phi_d, \phi_q) = \begin{cases} [\infty, 0] & \mathrm{Q}_{\text{avail}}(T_a, \text{Start}(M)) \leq 0 \\ [\mathbf{d}_0(M)/R^{\phi_d}(T_a, \text{Start}(M)), & \\ \mathbf{q}_0(M)R^{\phi_q}(T_a, \text{Start}(M))] & \mathrm{Q}_{\text{avail}}(T_a, \text{Start}(M)) > 0 \end{cases} \tag{3.7}$$

Causes. Task T_a causes method M_b, if the completion of the execution of T_a, in effect, also completes the execution of M_b (potentially an *action at a distance*; no explicit communication action is implied) and all properties of the completion of M_b ensue. More formally:

$$\text{causes}(T_a, M, t, d, q, \theta) = \begin{cases} [d, q] & t < \Theta(T_a, \theta) \\ [0, q] & t \geq \Theta(T_a, \theta) \end{cases} \tag{3.8}$$

Causal relationships are rare in the computational scheduling domains that most of our work has been in.

Shares-Results. The general class of facilitates relationships can also be used to model the effect of re-using partial results when switching between tasks (methods). This is modeled by the parameter $0 \leq \rho \leq 1$, the *retained time percentage*. When $\rho = 1$, switching tasks is like starting over completely; when $\rho = 0$, the new method is credited with all the time used under the original method. For example, assume we have methods M_a and M_b, with durations $\mathbf{d}(M_a)$ and $\mathbf{d}(M_b)$. Assume an agent starts using M_a and then switches to method M_b at time $t_1 < \mathbf{d}(M_a) + \text{Start}(M_a)$. We defined sharing results (in [Garvey and Lesser, 1993]) as follows:

$$\text{shares-results}(T_a, M, t, d, q, \rho) = \begin{cases} [d, q] & t < \text{Start}(M) \\ [d - (1-\rho)(\text{Start}(M) - \text{Start}(T_a)), q] & t \geq \text{Start}(M) \end{cases}$$

Note that the final quality is always limited by the maximum quality of the last method used (this definition could be expanded to hinder the following methods when going from a low-quality method to a higher-quality one; typically agents move from high- to low-quality, however). If there are several methods and the 'better' methods (with higher $\mathbf{q}(M)$) share results with 'worse' methods, then credit for time spent is cumulative.

Cancels. Informally, T_a cancels T_b if the completion of T_a, without any communication, causes $Q(T_b, t) = 0$ for times $t > \text{Finish}(T_a)$.

Favors. The favors relationship as described by von Martial [v. Martial, 1992] can be expressed as a combination of relationships already described. To put it simply, it is not really a new NLE (Non-Local Effect), but rather the presence of any positive (quality-improving) coordination relationship[11].

Von Martial describes a *favor action* predicate that holds for actions (in TÆMS, we call them methods): a *favor action* is an action which an agent does not have to execute locally and one that "the agent would be pleased to have executed by another agent" [v. Martial, 1992]. Von Martial then defines a favor relationship between two *plans* as holding when there exists some *favor action* a that is directly enabled by the plan actions, or can be enabled by adding a set of actions to the plan. Of course not all possible favor actions are worthwhile or cost effective! We can represent von Martial's concept in TÆMS as a relationship between a TÆMS method (the *favor action*) and a task (which represents a potential plan goal). A favors relationship between a method M and a task T holds when the method M can positively effect T (directly or indirectly) and M is executable by some agent beside the agent that has task T. That second agent then has a possibility of doing a 'favor' for the first agent by scheduling and executing M. How much of a 'favor' it is depends on the local utility of executing M, from the point of view of the agent offering the favor. M might be in that agent's current schedule, so the favor is essentially for free (except for communication and any necessary temporal synchronization, including a deadline on the completion of M). Alternately, M might not be in the current schedule, and a new schedule and a potentially new set of actions would be needed before and after executing M. Our definition is broader than von Martial's because the *favor action* M in our case does not have to be executable by both agents (only the agent who is doing the favor).

Figure 3.3 shows the canonical von Martial post office example. Assume that there are two agents and that each agent has a method for every task in the picture. Also assume that agent 1 wants to buy some stamps (task T1) and agent 2 wants to mail a package (task T2). The fact that agent 1 can execute a method for 'mail package', and the presence of the subtask relationship between 'mail package'[12] and task T2 means that agent 1 can do a 'favor' for agent 2 and do the 'mail package' task. Since agent 1 is already buying stamps, and thus already going to the post office and returning, such a favor might be fairly inexpensive. However, notice that packages must be mailed at the post office counter, while stamps can be bought at a self-service machine. If agent 1 is on a very tight schedule, it might not be possible to use the longer duration 'use counter' method. On the other hand, the same favor relationship exists from 'buy stamps' to task T1—agent 2 can also do a favor for agent 1 and buy the stamps. Since agent 2 is already committed to using the counter, no extra wait is involved.

Although I will not talk about the GPGP coordination mechanisms until Chapter 5, I will briefly discuss the implementation of a von Martial 'favors' coordination mechanism for cooperative agents. Such a mechanism would communicate the names of tasks on which agents want to achieve quality. Let's call these named tasks 'goals'. Agents then loop through each favor action (in von Martial's case, any action that can be executed by more than one agent; in our case the more general definition of any method or task that is related by a positive coordination relationship to another agent). If the favor action can positively affect the other

[11] Remember, a *coordination relationship* is an NLE between task structures at different agents.

[12] 'Mail package' is the *favor action* in the definition in the last paragraph, and subtask is the coordination relationship.

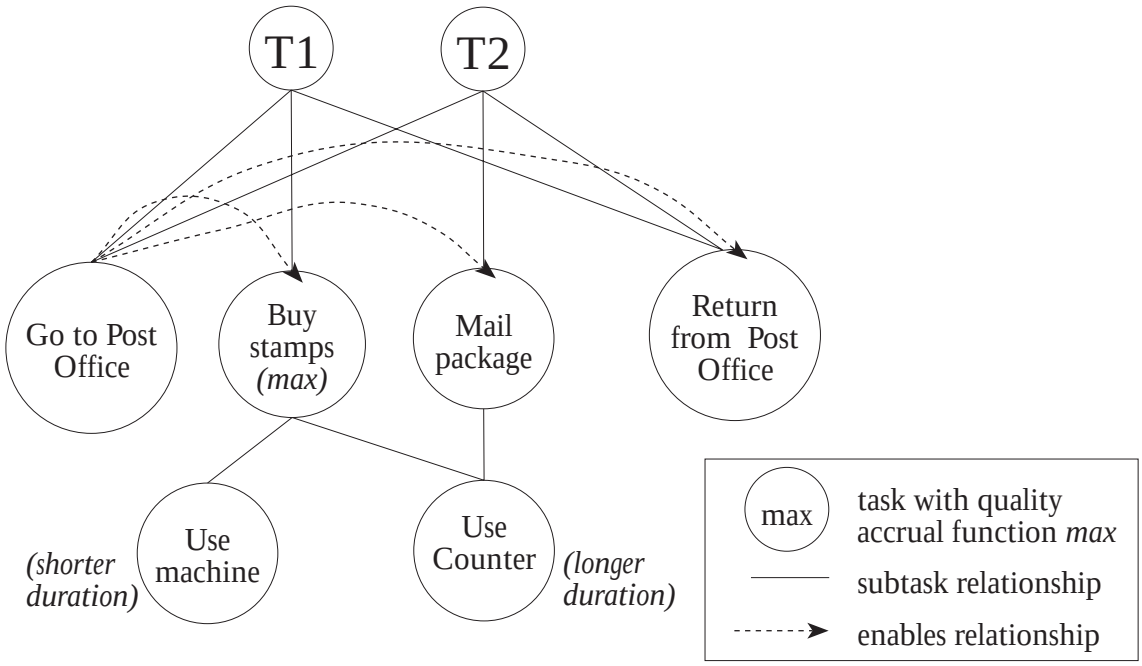

Figure 3.3. Objective task structure associated with visiting the post office to buy stamps and/or mail a package. Assume that every agent has a local method for each of these tasks.

agent's goals, then we mark it as a potential favor for that agent. This part of the algorithm is a straightforward restatement of von Martial's algorithm. At this point, von Martial would compute certain cost measures; in our case, we would query the local scheduler for the effect of adding a Do commitment on the favor action.

3.4.3.2 An Example of Facilitates and Shares-results

In this example we have one task group with two tasks, T_a and T_b, each with two methods, M_1 and M_2, and the non-local relationships indicated in Figure 3.4. For both tasks, $\mathbf{q}(M_1) = 20$ and $\mathbf{q}(M_2) = 5$, and $\mathbf{d}(M_1) = 15$ and $\mathbf{d}(M_2) = 5$. This example depends on our view of method execution at an agent that is discussed in Section 3.5.3.1.

This example uses the following assumptions:

- the execution of M_{1A} begins at time 0
- at time 4 the agent switches from method M_{1A} to M_{2A}
- communication of results from task T_a to task T_b occurs at the completion of M_{2A} (with no time delay)
- we use the linear quality model Q_{lin}
- method M_{2b} is scheduled for task T_b

Quality associated with each task and method then accrues over time as shown in Figure 3.5.

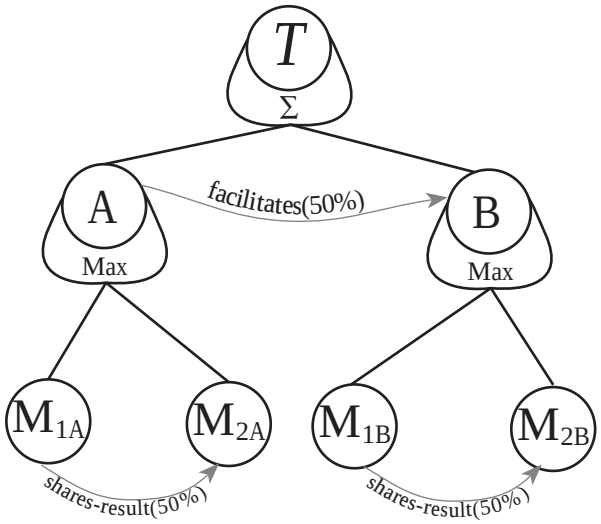

Figure 3.4. A simple model of two tasks with two methods each.

3.4.4 Expanding the DSN Model

We will now add some complexity to the distributed sensor network model. The length of a track l_i mentioned earlier is a generative model parameter. Given a set of these generative parameters, we can construct the objective model for a specific episode (problem-solving instance). Figure 3.2 shows the general structure of episodes in our DSN environment model. To display an actual objective model, let us assume a simple, concrete situation: there are two agents, A and B, and that there is one vehicle track of length 3 sensed once by A alone (T^1), once by both A and B (T^2), and once by B alone (T^3). We now proceed to model the standard features that have appeared in our DVMT work for the past several years. We will add the characteristic that each agent has two methods with which to deal with sensed data: a normal VLM and a 'level-hopping' (LH) VLM (the level-hopping VLM produces less quality than the full method but requires less time; see [Decker *et al.*, 1990, Decker *et al.*, 1993b] for this and other approximate methods; a similar technique can be used to model agents who have different capabilities such as processor speed). Furthermore, only the agent that senses the data can execute the associated VLM; but any agent can execute VTMs and VCMs if the appropriate enablement conditions are met.

Figure 3.6 displays this particular problem-solving episode. To the description above, we have added the fact that agent B has a faulty sensor (the durations of the grayed methods will be longer than normal); we will explore the implications of this after we have discussed the subjective level of the framework in the next section. An assumption made in [Durfee *et al.*, 1987] is that redundant work is not generally useful; this is indicated by using *max* as the combination function for each agent's redundant methods. This would *not* be a good assumption for an environment where the independent derivation of a result is a *positive* thing—we can alter the assumption by simply changing this function (to *mean* or *sum*). Another characteristic that appeared often in the DVMT literature is the sharing of results between methods (at a single agent); we would indicate this by the presence of a sharing relationship (similar to facilitates) between each pair of normal and level-hopping VLMs

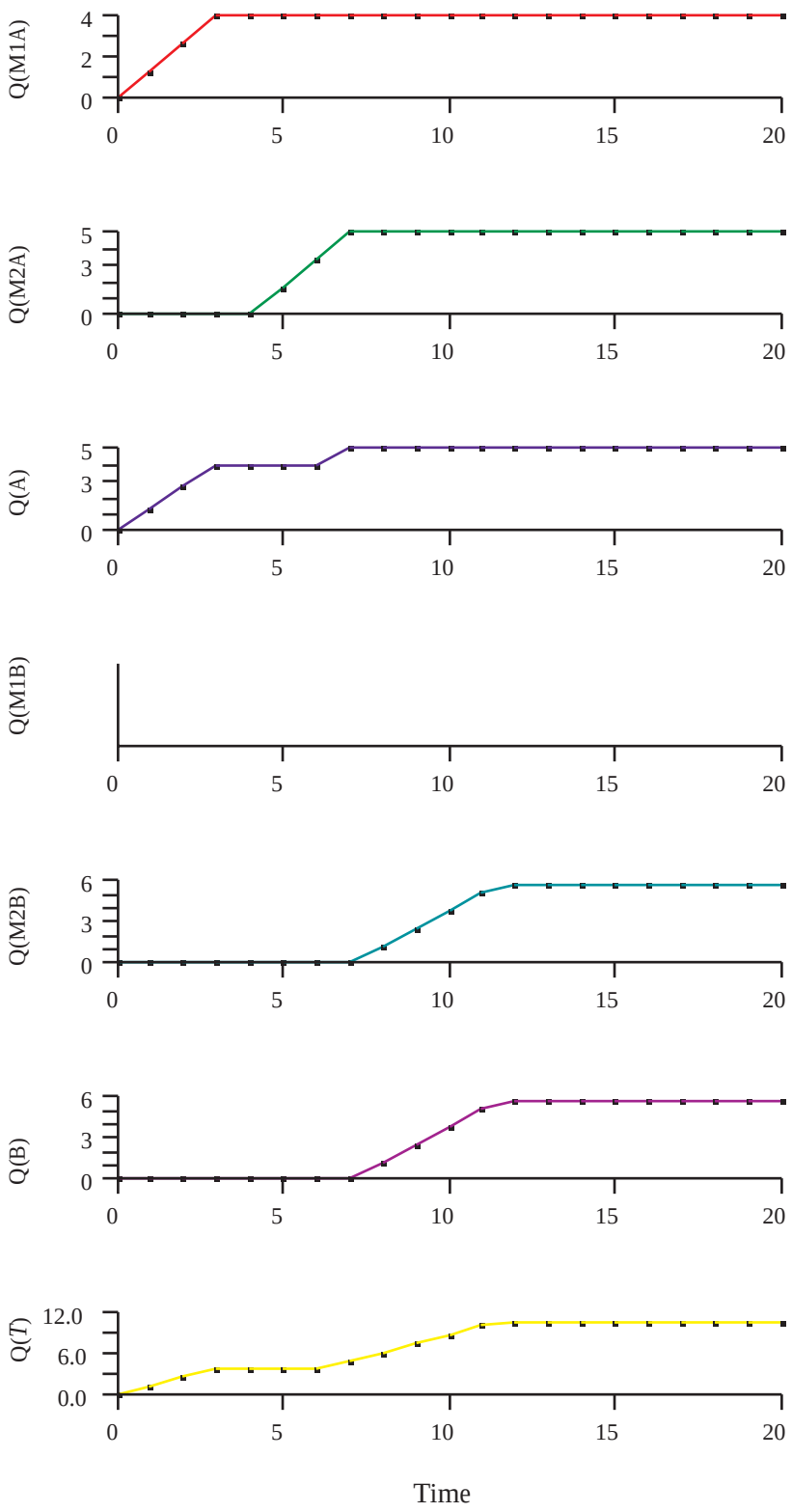

Figure 3.5. Quality at each Method, Task, and Task Group over time in the previous figure

(note: this is not indicated in the figure). Sharing of results could be only one-way between methods.

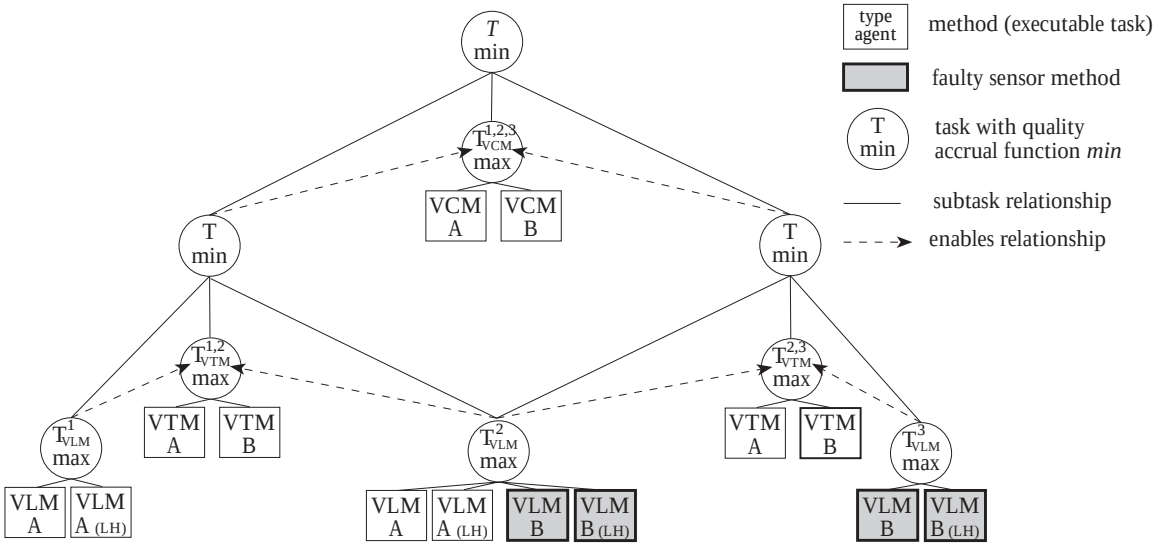

Figure 3.6. Objective task structure associated with two agents

Now we will add two final features that make this model more like the DVMT. First, low quality results tend to make things harder to process at higher levels. For example, the impact of using the level-hopping VLM is not just that its quality is lower, but also that it affects the quality and duration of the VTM it enables (because not enough possible solutions are eliminated). To model this, we will use the precedence relationship (precedes) instead of enables: not only do the VLM methods enable the VTM, but they can also hinder its execution if the enabling results are of low quality. Secondly, the first VLM execution provides information that slightly shortens the executions of other VLMs in the same vehicle track (because the sensors have been properly configured with the correct signal processing algorithm parameters with which to sense that particular vehicle). A similar facilitates effect occurs at the tracking level. These effects occur both locally and when results are shared between agents—in fact, this effect is very important in motivating the agent behavior where one agent sends preliminary results to another agent with bad sensor data to help the receiving agent in disambiguating that data. Figure 3.7 repeats the objective task structure from the previous figure, but omits the methods for clarity. Two new tasks have been added to model facilitation at the vehicle location and vehicle track level.[13] T_{VL} indicates the highest quality initial work that has been done at the vehicle level, and thus uses the quality accrual function *maximum*. T_{VT} indicates the progress on the full track; it uses *summation* as its quality accrual function.

[13]Note that these tasks were added to make the model more expressive; they are not associated with new methods.

The more tracking methods are executed, the easier the remaining ones become.[14] This model has implications for a multi-agent episode. For instance, the model describes the effect of the communication time on communicating partial results to another agent: the later an agent delays communication, the more the potential impact on the other agent (because the quality is increasing), but the more the other agent must delay. We examine this question in somewhat more detail in Chapters 4, 5, and 6.

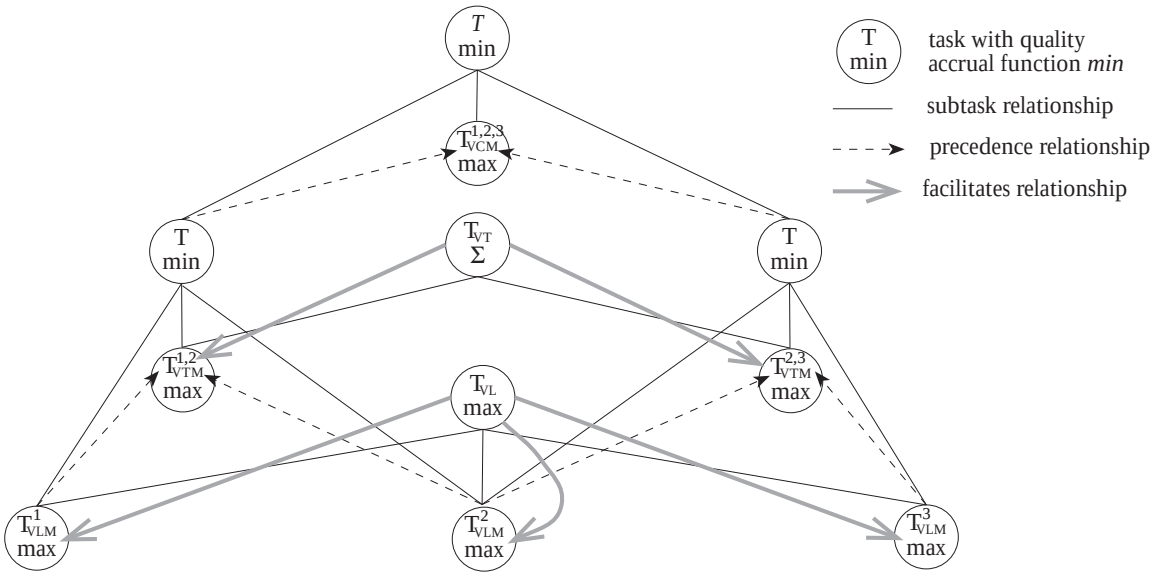

Figure 3.7. Non-local effects in the objective task structure

At the end of the next section, we will return to this example and add to it subjective features: what information is available to agents, when, and at what cost.

3.4.5 Resources

Physical resources, be they people, equipment, or computational resources such as files, are very important to model in many domains [Zhao *et al.*, 1987, Cheng *et al.*, 1988, Ramamritham *et al.*, 1990, Shen *et al.*, 1993, Sycara *et al.*, 1991, Neiman *et al.*, 1994]. This dissertation is mainly concerned with the modeling of computational task structures that do not use external physical resources (physical resources other than the processor). This section will briefly describe how we could extend TÆMS to model physical resources. The rest of the dissertation will concentrate on purely computational environments—we will not carry out any simulation experiments with physical resources in this dissertation. We believe this is an important future direction, see the Future Work section in Chapter 7.

[14] In the real DVMT, the effect only works to a point, so that the first four data points or so bring about most of the effect. We could model this more precisely by replacing the summation with a Max and a Sum (a piecewise linear approximation) or by using an exponential ($\frac{1}{1-e^{s}}$).

We have recently extended TÆMS so that resources can be represented directly in task structures. From the point of view of task environment specification, the only effect of resources, with respect to computational tasks, is to change the duration or quality of other tasks, and so we represent this as a set of non-local effects (NLE's). Because the NLEs are linked (i.e., not just between two nodes) we add a *resource* node to the task structure and associate any state information needed with it. From the point of view of the discrete, state-based mathematics behind TÆMS(summarized in Section 3.2), a *resource* looks like a task—but instead of a quality vector, a nominal state vector is used, and there is no associated duration or subtask relationships. This idea is very similar to the way we added tasks to represent the amount of work done on the first vehicle location and the vehicle track in Figures 3.6 and 3.7 in Section 3.3.

Instead, methods are related to the resource via one kind of NLE (e.g. uses, replenishes, consumes, reads, writes), and another NLE runs from the resource to the methods (e.g. exclusiveaccess, limitedbandwidth, consumable). The method-to-resource NLE's change the state of the resource, and the resource-to-method NLE's affect duration and max quality as usual.

For example, take the situation of a low-bandwidth communication link as described in [Sugawara and Lesser, 1993]. Two agents have a diagnosis method that uses the low-bandwidth link. When more than one of these diagnosis methods are executed at the same time, the link is saturated and the durations of the methods are lengthened. We represent this situation as shown in Figure 3.8; the two new NLE's are defined as follows:

$$\mathsf{uses}(T_a, R, t, d, q, \alpha) = \begin{cases} [d, q+\alpha] & \mathrm{Start}(T_a) < t < \mathrm{Finish}(T_a) \\ [d, q] & \text{otherwise} \end{cases} \tag{3.9}$$

$$\mathsf{limits}(R, M, t, d, q, \sigma, \phi_d) = \begin{cases} [\phi_d d, q] & Q(R,t) > \sigma \\ [d, q] & \text{otherwise} \end{cases} \tag{3.10}$$

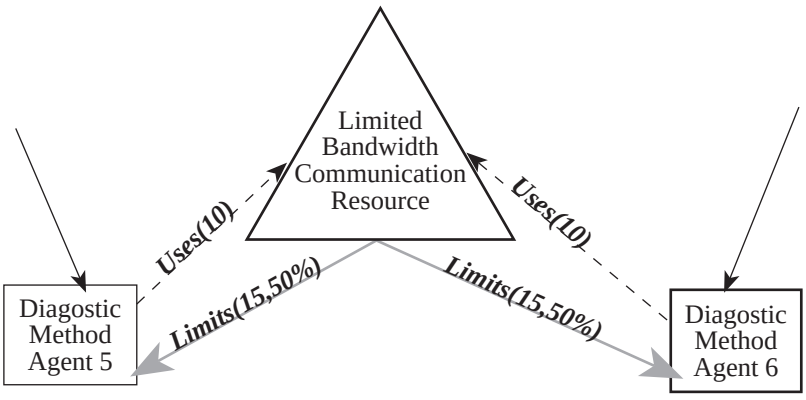

Figure 3.8. Example of two methods sharing a limited resource of capacity 15. In this example, if both methods execute in temporally overlapping time periods, the durations of each method will be lengthened by 50%.

The NLE uses indicates the amount of limited resource capacity used by the method with the parameter α. In Figure 3.8 each diagnostic method uses 10 units of the communication resource's capacity (the arc labeled $\mathsf{uses}(10)$). The NLE limits indicates that the resource has maximum capacity (saturation) σ and that beyond that point duration is affected by percentage ϕ_d (alternately, ϕ_d could be defined as a function of the level of oversaturation). In Figure 3.8 the communication resource has a saturation point $\sigma = 15$, so the resource will not be saturated if either diagnostic method uses the resource alone, and will be saturated if the two methods happen to overlap in their execution. The effect will only be active during the overlap, and will cause the duration of both methods to be increased (in this example) by 50%. Note that when methods can be interrupted, the lengthening of method durations due to a blocked resource is not associated with lengthening the amount of continuous computation—the blocked method can be interrupted and other computations performed until the method is no longer blocked.

It is important to note that our purpose here is not to develop a full-fledged resource representation language, but merely to argue that we could do so at a later time (see Future Work in Chapter 7). The extensibility of TÆMS, including the addition of new task and resource relationships, is one of its major advantages. The exciting thing is that at the lowest level, we can define resource effects without adding to the basic, simple TÆMS mechanisms. Note that the formulae above and below use the Quality variable to hold state information for the resource; if a resource requires more than one state variable then we can represent it by using a quality vector, as we noted at the start of this chapter.

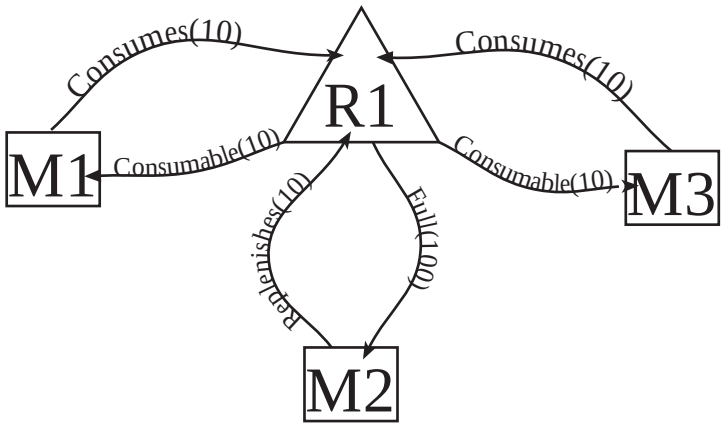

Figure 3.9. Example objective structure of two methods (M1 and M3) that consume a resource, and one method that replenishes it.

Similar NLE's can be defined for representing consumable resources and their associated operations (see Figure 3.9):

$$\mathsf{consumes}(T_a, R, t, d, q, \alpha) = \begin{cases} [d, q - \alpha] & t > \mathrm{Finish}(T_a) \\ [d, q] & \text{otherwise} \end{cases} \tag{3.11}$$

$$\mathsf{consumable}(R, M, t, d, q, \alpha) = \begin{cases} [\infty, 0] & Q(R, t) < \alpha \\ [d, q] & \text{otherwise} \end{cases} \tag{3.12}$$

$$\mathsf{replenishes}(T_a, R, t, d, q, \alpha) = \begin{cases} [d, q+\alpha] & t > \mathrm{Finish}(T_a) \\ [d, q] & \text{otherwise} \end{cases} \tag{3.13}$$

$$\mathsf{full}(R, M, t, d, q, \sigma) = \begin{cases} [\infty, 0] & Q(R,t) > \sigma \\ [d, q] & \text{otherwise} \end{cases} \tag{3.14}$$

Thus in Figure 3.9 Methods 1 and 2 consume 10 units of Resource 1, if that amount is available and Method 3 when executed replenishes Resource 1 by 10 units, unless Resource 1 already has over 100 units, in which case it cannot be replenished (it is 'full'). Note that this is an *objective* example. A generative model would (for example) say if Method 1 always consumes 10 units or Resource 1, or if the amount is drawn from some distribution. The subjective model would say if an agent, making a decision about Method 1, "knows" that Method 1, in fact, consumes 10 units, or believes that it consumes more or less units, or knows only the generative model.

3.5 TÆMS Subjective Level Models and Agent Actions

The purpose of a subjective level model of an environment is to describe what portions of the objective model of the situation (current state) are available to 'agents'. It answers questions such as "when is a piece of information available," "to whom is it available," and "what is the cost to the agent of that piece of information". This is a description of how agents might interact with their environment—what options are available to them. Of course agents both acquire information and *act* on that information. This chapter will discuss agent actions—acquiring information, executing methods, and communicating with one another.

To build such a description we must introduce the concept of *agency* into the model. Ours is one of the few comprehensive descriptions of computational task environments in AI, but there are many formal and informal descriptions of the concept of agency (see [Gasser, 1991, Hewitt, 1991, Moe, 1984], Sections 2.3.3 and 2.4.4). Rather than add our own description, we notice that these formulations define the notion of *computation* at one or more agents, not the environment that the agents are part of. Most formulations contain a notion of *belief* that can be applied to our concept of "what information an agent has about its environment". Our view is that an "agent" is a locus of belief and action (such as computation), as stated in the introduction to this chapter.

The form of the rest of this section is as follows:

- How does the environment affect the beliefs of the agents? This is described by a TÆMS "subjective" level model.
- How do the beliefs of agents affect their actions? This is basis of the coordination and control problem. The rest of the dissertation will describe and analyze various mechanisms to accomplish this step.
- How do the actions of the agents affect the environment? /tems/ is a discrete, state-based model. We will describe in this section how the state of the environment and the agents changes when the agents act. There are three types of agent actions: method executions, communications, and information gathering actions.

3.5.1 The Subjective Mapping

Each agent can be thought of to have a "belief database" Γ_A. We use the symbol Γ_A^t to denote this set of beliefs of agent A at time t. We will use the modal operator $B_A^t(x)$ to mean agent A subjectively believes x at time t, that is, $\Gamma_A^t \models x$ (from Shoham[Shoham, 1991]).

We define a *subjective mapping* $\varphi : [(x \in \mathbf{E}) \times A \times t] \mapsto x'$ that maps from elements in the current episode to agent A's subjective view of those elements. The mapping may be empty for an element. This is the core of the subjective level model. For example, we could define a mapping φ where each agent has a probability p of believing that the maximum quality of a method is the objective value, and a probability $1 - p$ of believing the maximum quality is twice the objective value. Any objective assertion has some subjective mapping, including $\mathbf{q}$ (maximum quality of a method), $\mathbf{d}$ (duration of a method), deadlines, and the relations subtask, nle, and comm and their associated parameters. Section 3.5.4 will describe a few examples of this mapping.

Finally, we will define an action that can be taken by an agent, called an *information gathering action*, which will take some amount of time and result in the addition of new information to the agent's belief database. In particular, we define a *task arrival information gathering action* such that

$$\Gamma_A \leftarrow; \Gamma_A \cup \varphi(x, A, t) | (x \in \mathbf{E}) \wedge (\varphi(x, A, t) \notin \Gamma_A)$$

This action places a subjective version of new information from the objective episode into the agent's belief database.

3.5.2 Deciding What to Do Next: Coordination, Scheduling, and Control

The beliefs of an agent affect its actions through some control mechanism. Since this is the focus of most of our and others' research on local scheduling, coordination, and other control issues, we will not discuss this further in this chapter (but see Chapters 4 through 6). The agent's control mechanism uses the agent's current set of beliefs Γ_A to update three special subsets of these beliefs $\mathcal{I}, \mathcal{C}, \mathcal{M} \subset \Gamma_A$ defined here as the sets of information gathering, communication, and method execution actions to be computed. These subsets describe the agent's *intentions*. For example, if an agent intends to execute method M1, it creates a method execution action that indicates M1 as the method to execute.[15] The agent then adds this method execution action set of intended method executions $\mathcal{M}$. The next section describes what happens to an agent's intended actions.

The agents that we build typically further divide control into *local scheduling* and *coordination* (see Chapters 5 and 6), but this is not required. Besides describing how an agent's beliefs entail commitments to particular information gathering, communication, and method execution actions, a control component model must also describe the *duration* of its deliberations. This feature allows us to analyze questions concerning the cost of control without becoming mired in implementation details.[16]

[15] It might indicate other things, like on which processor to execute M1 if the agent is a multiprocessor, or a time limit for M1, execution monitoring information, etc. None of these extensions are a part of this dissertation.

[16] Understanding the details of the control costs of particular algorithm implementations is important, but usually not at early stages of research. Information about control duration can also be used by the TÆMS simulator if it is available.

3.5.3 Computation

Our model can support parallel computation, but for brevity we will just describe single processor computation as a sequence of agent states. Agent A's current state is uniquely specified by Γ_A. We provide a meta-structure for the agent's state-transition function that is divided into the following 4 parts (see also Figure 3.10:

1. control (as described in the previous section—basically asserting one or more intentions)
2. method execution (of the methods in $\mathcal{M}$)
3. information gathering (doing the information gathering actions in $\mathcal{I}$)
4. communication (doing the communication actions in $\mathcal{C}$)

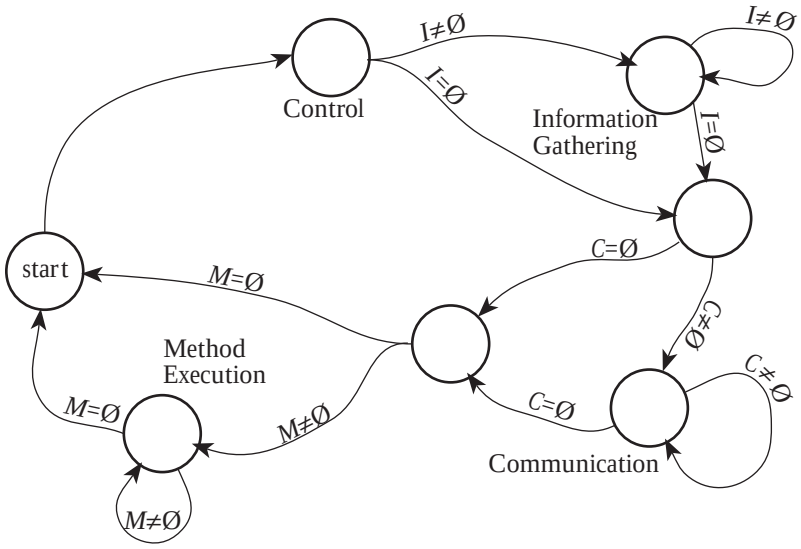

Figure 3.10. The finite state machine that describes the meta-structure of single-processor agent computations. $\mathcal{I}, \mathcal{C}, \mathcal{M}$ are named subsets of the agents beliefs Γ that represent *information gathering*, *communication*, and *method execution* actions, respectively.

First the control mechanisms assert ("intend") information-gathering, communication, and method execution actions and then these actions are computed one at a time, after which the cycle of meta-states repeats.

A simple model of parallel computation, similar to the implementation in [Decker *et al.*, 1993a], is to allow control, information gathering, and communication to run on one (abstract) processor, and multiple method executions on the other processors. Any important interactions between methods executing in parallel would be represented by non-local effects.

3.5.3.1 Method Execution

How do the actions of an agent affect the environment? Both the objective environment (e.g. quality of the executing method) and the subjective mapping (e.g. information available via φ) can be affected. We use two execution models: simple method execution, and execution with monitoring, suspension, and preemption. These follow from the discussion of Q_{DTT} and Q_{lin} in Section 3.4.2, and are simple state-based models. Basically, for non-interruptible, single processor method executions, the agent enters a method execution state for method M at time $\mathrm{Start}(M)$ and remains in that state until the time t when $t - \mathrm{Start}(M) = \mathbf{d}(M, t)$. Method execution actions are similar to what Shoham terms 'private actions' like DO[Shoham, 1991].

We have also considered pre-emptable method execution, where a method execution action is given a set upper time limit, after which computation will proceed to the next meta-state. The agent can then monitor the execution of long methods, and interleave their execution with other actions or pre-empt them entirely [Garvey and Lesser, 1993].

Formal Definition of Simple Method Execution. Let $M \in \mathcal{M}$ be the method that an agent is about to execute. We identify several parts of the objective model $\mathcal{O}$ of the external environment: $\mathbf{q}(M, t)$, the true maximum quality of M at time t computed from some initial $\mathbf{q}_0(M)$ and all of the non-local effects on M as described in Section 3.4.3; $\mathbf{d}(M, t)$, the true duration of M at time t computed from some initial $\mathbf{d}_0(M)$ and all of the non-local effects on M as described in Section 3.4.3; and Q, the mathematical model of the way quality accrues during the execution of M (Q_{DTT}, for example).

At the start of execution at time t_0, the state of the environment is updated by an agent A such that $\langle \mathrm{Start}(M) = t_0 \rangle \in \mathcal{O}$. The state of the environment includes the values of $\mathbf{q}(M, t_0)$ and $\mathbf{d}(M, t_0)$ computed as described in Section 3.4.3. Agent A at time t_0 is in a state where it believes $B_A^{t_0} \langle \mathrm{Start}(M) = t_0, \mathbf{q}(M, t0) = \varphi(\mathbf{q}(M, t_0)), \mathbf{d}(M, t0) = \varphi(\mathbf{d}(M, t_0)) \rangle$. Depending on the model of quality accrual Q, the quality of M at time $t + 1$ may change, which may trigger non-local effects, etc.

From its state at an arbitrary time t the environment moves to a new state at time $t + 1$ by updating the values of $\mathbf{q}$ and $\mathbf{d}$ for all M, given the addition of new start times $\mathrm{Start}(M)$ for methods as agents execute them, and new finish times $\mathrm{Finish}(M)$ as agents finish execution (described next). The agent also moves to a new state at time $t + 1$: if $t+1 < \mathrm{Start}(M) + \mathbf{d}(M, t)$[17] then the agent remains in the state where it believes $B_A^{t+1} = B_A^t$, otherwise it finishes execution, and adds to its beliefs and the objective model the assertion $\mathrm{Finish}(M) = t + 1$.

The simple model of execution does not allow agents to preempt, suspend, or otherwise modify the execution of a single method. This capability can be added by extending this model so that a method execution intention is a pair $[M, t] = \mathcal{M}$ which means to execute M for t time units, or until it finishes, if it finishes early. This simple extension allows both monitoring and preemption, and improves the model because the local scheduler can intend information-gathering actions (i.e., see if new tasks have arrived) at the monitoring points.

3.5.3.2 Communication

How do the actions of an agent affect other agents? Communication actions allow agents to affect each others' beliefs to a limited extent. Many people have worked on formalizing

[17] Note that $\mathbf{d}(M, t)$ is the actual objective duration of M at time t, not the agent's belief in its duration.

aspects of communication; the semantics of communication actions can be freely defined for each environment. The simplest communication act is to send another agent the 'current result/value' of a method—the effect is to change the available quality $Q_{\text{avail}}(T, t, A)$ at the remote agent after the message has been received. What happens when a communication is 'received'? The reception of information, by changing the available quality of a task, may trigger a non-local effect as we described earlier, and may influence the behavior of an agent as specified by its control algorithm. Communication actions may take time themselves (defined in a TÆMS model) and there may be a delay in communication between agents (as discussed in section 3.4.3).

3.5.3.3 Information Gathering

An information gathering action trades-off computational resources (time that could be spent executing methods) for information about the environment. For example, one useful information gathering action is the task arrival information gathering action we described in Section 3.5.1 that queries the environment about the arrival of new tasks or task groups. Another information gathering action causes any communications that have arrived at an agent to be 'received' (added to the agent's belief database). A third kind of information gathering may identify coordination relationships—non local effects that span multiple agents (see Chapter 5). Both communication and information gathering actions take some period of time (not necessarily constant) to execute, as specified in the model. A fourth type of information gathering might be a call to a 'planner'.

3.5.4 Subjective Modeling Example

Let's return to the example we began in Section 3.3 to demonstrate how adding a subjective level to the model allows us to represent the effects of faulty sensors in the DVMT. We will define the default subjective mapping to simply return the objective value, i.e., agents will believe the true objective quality and duration of methods and their local and non-local effects. We then alter this default for the case of faulty (i.e., noisy) sensors—an agent with a faulty sensor will not initially realize it ($\mathbf{d}_0(\text{faulty-VLM}) = 2\mathbf{d}_0(\text{VLM})$, but $\varphi(A, \mathbf{d}_0(\text{faulty-VLM})) = \mathbf{d}_0(\text{VLM})$).[18] Other subjective level artifacts that are seen in [Durfee *et al.*, 1987] and other DVMT work can also be modeled easily in our framework. For example, 'noise' can be viewed as VLM methods that are subjectively believed to have a non-zero maximum quality ($\varphi(A, \mathbf{q}_0(\text{noise-VLM})) > 0$) but in fact have 0 objective maximum quality, which the agent does not discover until after the method is executed. The strength with which initial data is sensed can be modeled by lowering the subjectively perceived value of the maximum quality $\mathbf{q}$ for weakly sensed data. The infamous 'ghost track' is a *subjectively* complete task group appearing to an agent as an actual vehicle track, which *subjectively* accrues quality until the hapless agent executes the VCM method, at which point the true (zero) quality becomes known. If the track (subjectively) spans multiple agents' sensor regions, the agent can potentially identify the chimeric track through communication with the other agents, which may have no belief in such a track (but sometimes more than one agent suffers the same delusion). In general,

[18] At this point, one should be imagining an agent controller for this environment that notices when a VLM method takes unusually long, and realizes that the sensor is faulty and re-plans accordingly.

having different subjective mappings for different agents or classes of agents allows us to model situations where some agents are more, less, or simply differently 'informed' than others.

3.5.5 Sugawara's Network Diagnosis System

Let us briefly return to Sugawara's network diagnosis system, as introduced in Section 3.4.5 and figure 3.8. Sugawara's system makes decisions about how to schedule and coordinate diagnosis tasks based on its subjective, perceived view of the environment. This subjective view may be wrong. The diagnosis system monitors its own progress and when progress does not match its subjective expectations, it may generate an explanation of this discrepancy and update its subjective view. The new subjective view may result in tasks being scheduled or coordinated differently. For example, Figure 3.11 shows a subjective view where the resource is essentially unlimited—this situation results in no coordination when executing the diagnostic methods. However, if the real objective situation is the one on the lower right, a problem will occur if the two methods are executed simultaneously and the diagnosis system will spot this. Information gathering actions will then be added to detect which situation is occurring and coordination actions will be added to the schedule to avoid overloading the bottleneck resource if necessary.

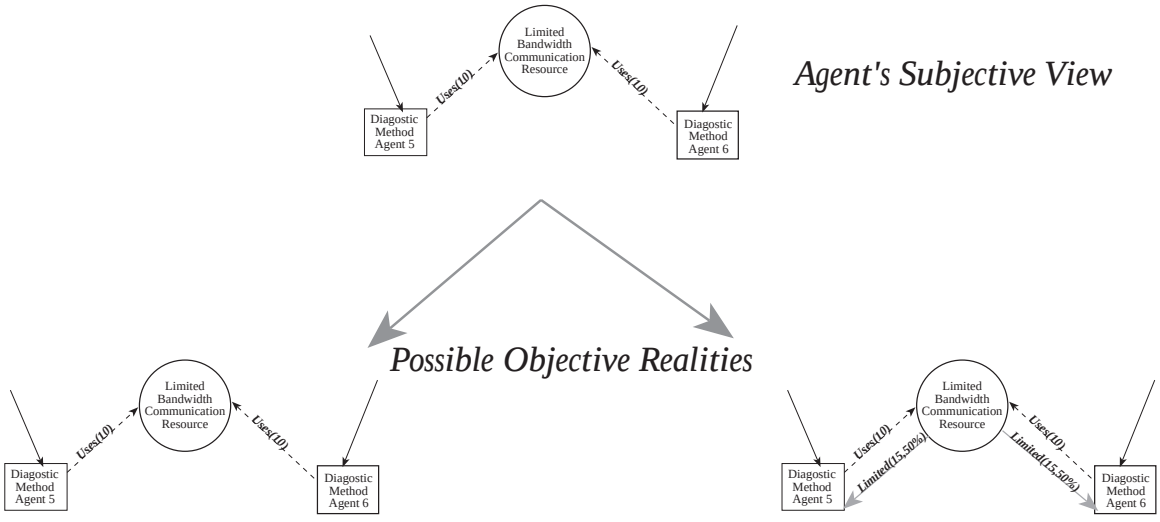

Figure 3.11. Example of what is learned in the network diagnosis problem: the correct scheduling strategy for each objective situation, and the knowledge of which situation is currently occurring.

We have explained the objective and subjective levels of our modeling framework, and presented an example of a moderately complex task structure. Next we turn to generative level models, where we specify the statistical properties of an environment across many episodes.

3.6 TÆMS Generative Level Models and Framework Examples

3.6.1 The Generative Level

By using the objective and subjective levels of TÆMS we can model any *individual* situation; adding a *generative* level model allows us to go beyond that and determine what the expected performance of an algorithm is over a long period of time and many individual problem solving episodes. A generative-level model can be viewed as a workload (episode) generator for an environment. In our work we have used statistical generative models of task inter-arrival times (exponential distribution), amount of work in a task group (Poisson), task durations (exponential), and the likelihood of a particular non-local effect between two tasks (we'll see more of this in Chapters 5 and 6 [Decker and Lesser, 1993b, Decker and Lesser, 1993a, Garvey and Lesser, 1993]. Generative level statistical parameters can also be used by agents in their subjective reasoning. For example, an agent may make control decisions based on its knowledge of the expected duration of methods.

A generative level model can be constructed by careful analysis of the real environment being modeled, or by observing the statistical properties of real episodes (if that is possible). Even when certain parameters of the real world are unknown, they can be made variables in the model and then you can ask questions about how much they affect the things you care about. Our approach so far has been to verify our assumptions about the environment with simple statistical approaches [Kleijnen, 1987]. Detailed model verification will be more important when using our framework to optimize parameters in a real application, which is not a focus of this dissertation. In this dissertation we will focus on learning the general effects of parameters on a coordination algorithm (see Chapters 5 and 6).

An excellent way to proceed in building a generative model (the one we have used) is to gather several detailed TÆMS objective and/or subjective task structures for specific episodes. These can be then examined to determine the obvious endogenous features of the episode that are provided by the environment. For example, the number and lengths of tracks in DSNs; the set of tests ordered for patients in a hospital scheduling problem; arrival lateness and resource failure rates for an airport resource management problem. Developing a task structure for several episodes constrains and focuses the model-builder. Future work could examine the possibility of inducing generative parameters from the observation of episodic task structures.

In our DSN model example, any single episode can be specified by listing the task groups, and what part of each task group was available to which agents, given the organizational structure. Our analyses are based on the statistical properties of episodes in this environment, not any single instance of an episode. The properties of the episodes in a DSN environment are summarized by the tuple $\mathcal{D} =< A, \eta, r, o, \mathcal{T}(l) >$ where A specifies the number of agents, η the expected number of task groups, r and o specify the structural portion of the organization by the *range* of each agent and the *overlap* between agents[19], and $\mathcal{T}(l)$ specifies the homogeneous task group structure (as discussed in Sections 3.3 and 3.5.4). A particular episode in this environment can be described by the tuple $\mathbf{D} =< \mathcal{T}_1(l_1), \ldots, \mathcal{T}_n(l_n) >$ where n is a random variable drawn from a Poisson distribution with an expected value of η, and the l_i are generated as described in Section 3.3. If we were to extend this generative model to cover every feature we added to the objective and subjective models in Sections 3.3 and 3.5.4, we would need to add the likelihood of a sensor being faulty (noisy), the likelihood of a ghost track, etc.

[19] We also assume the agents start in a square geometry, i.e, 4 agents in a 2×2 square, 25 agents arranged 5×5.

3.6.2 The TÆMS Simulator Generator

In order to test coordination and scheduling algorithms in general environments, the TÆMS simulator has a random task structure generator. Describing this generator will give you an example of how generative-level models of abstract task environments can be constructed. This generator does *not* generate structures from a "real" application environment; the structures it generates are quite abstract. When working with TÆMS on a particular application (such as in the DSN problem in the next chapter), we do not use this random generator, but rather one crafted to match the specific environment. These random, abstract environments are useful for general experimentation and performance analysis of scheduling and coordination algorithms.

The random environment generator has three parts: specifying general environmental parameters such as the inter-agent communication delay, specifying a generative subjective template, and specifying the potential classes of task groups (generative objective templates).

The default generative subjective template takes a list of agent names and a simple probability p_{overlaps} for the chance of overlapping methods. For each objective method generated and for each other agent, there is a p_{overlaps} chance that the agent will have an overlapping method. All objective values are passed unchanged to the agents. Agents who execute information-gathering actions will receive information on all methods executable by them, and all parents of those methods, recursively. The default generative subjective template guarantees no other subjective properties.

A random environment will consist of one generative subjective template and a list of generative objective templates (one for each potential class of task groups). Each generative objective template takes the following parameters:

branching-factor: The average number of children of a task in the task structure. Specified as the mean (equals the variance) of a Poisson distribution.

depth: The average depth of the task structure (not counting the leaf methods). Specified as the mean (equals the variance) of a Poisson distribution.

duration: The average initial duration ($\mathbf{d_0}$) of a method. Specified as the mean of an exponential distribution.

max-quality: The average initial max-quality ($\mathbf{q_0}$) of a method. Specified as the mean of an exponential distribution.

QAF: The distribution of potential Quality Accumulation Functions for tasks. Specified as a custom discrete distribution such as "50% Min, 50% Max".

Deadline: Deadline tightness. The deadline for the task group is set to αESAET $+ \beta$ where ESAET is the Estimated Single Agent Execution Time of the actual instantiated task group, and α and β are the specified parameters. The calculation of ESAET is only an estimate because it ignores non-local effects.

Inter-Arrival-Time: The mean time between arrivals of instances of this task group class. Specified as the mean of an exponential distribution.

Redundant-QAF: The Quality Accumulation Function to use when combining overlapping methods. If the methods are redundant, then MAX is an appropriate choice.

The TÆMS random structure generator also allows the specifications of certain 'patterns' of relationships on top of this basic structure. These patterns have parameters that are in addition to the generative objective template parameters just mentioned:

Random-Relations: Generates hard and soft relationships with some probability but in no particular pattern. Hard relationships are always generated across siblings to prevent loops (i.e., pick a sibling, test if it should be hard-related to any of the remaining siblings, then pick one of the remaining siblings and continue until no siblings are left). The random-relations pattern takes two new parameters:

hard-rels: A discrete distribution of potential relationships or NIL for no relationship. For example, "10% enables, 90% NIL".

soft-rels: A discrete distribution of potential relationships or NIL for no relationship. For example, "20% facilitates, 10% hinders, 70% NIL".

cleanup-methods: One possible environmental pattern (seen in the DVMT) is the presence of a method that must be done at the completion of other subtasks of an AND task. The VTM's and VCM in Figure 3.2 are a few examples. This pattern generates such 'cleanup methods' that are subtasks to an AND task and that are enabled by all the other subtasks of the AND task. The cleanup-methods pattern takes one new parameter:

cleanup-prob: The simple probability that such a method is actually generated.

Fast-fallback-at-leaves: In environments where design-to-time scheduling is useful, there are often multiple ways of achieving some task that do not all have to be done. A quick-and-dirty method that can be executed when the agent is under severe time pressure is called a 'fast fallback method'. This pattern generates such methods as alternatives to existing 'regular' methods at the leaves of the task structure. The fast-fallback-at-leaves pattern takes three new parameters:

fast-fallback-prob: The probability that a given method will have a fast-fallback alternative.

fast-fallback-percentage: A specification of the time/quality tradeoff represented by the fast fallback method (how much faster and how much less quality).

fast-fallback-hinders-p: Often doing something the quick and dirty way will end up hurting you later on (for example, the hindering precedence relationship in the DVMT structure of Figure 3.7). This parameter, if not NIL, specifies the parameters to a hinders relationship between the the fast fallback method and its parent task's parent task.

3.6.3 Examples

In this section I will present several examples of TÆMS models for several quite different environments. This will demonstrate the flexibility of the approach, and allow me to discuss the impact of each task environment on coordination issues (even with relatively sketchy models). The distributed sensor network model, which has been the focus of the examples in the last three chapters, will not be repeated here.

3.6.3.1 Hospital Patient Scheduling

This description is from an actual case study [Ow *et al.*, 1989]:

> *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.*

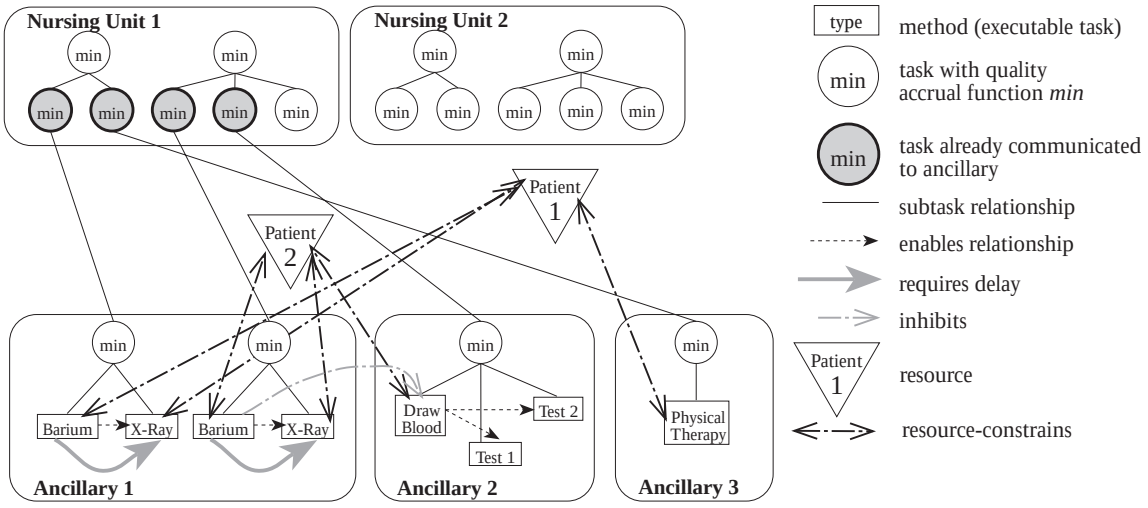

Figure 3.12. High-level, objective task structure and subjective views for a typical hospital patient scheduling episode. The top task in each ancillary is really the same objective entity as the unit task it is linked to in the diagram.

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.

A generative level model of this environment would focus on the mean time between patient arrivals, and the number and distribution of patient tests. Other generative parameters would include the specification of the 'agents' in this environment: the number of nursing units and the number and type of ancillaries and their associated resources (i.e., how many X-ray machines can be used in parallel by ancillary 1). Thus an episode generator for Ow's hospital would have a fixed number of nursing units and ancillaries, and a fixed set of test templates corresponding to taking X-rays, physical therapy, blood tests, etc. These templates would also

contain duration distributions on how long it takes to take the x-ray, etc. Then, we would postulate an arrival rate for patients, and for each patient a distribution of tests (instantiated from the set of templates). Thus when each patient arrives, linked to that patient is a unique task structure of what needs to be done to that patient.

Figure 3.12 shows a hand-generated example objective 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. I use *min* (AND) to represent quality accrual because in general neither the nursing units nor ancillaries can change the doctor's orders—all tests must be done as prescribed. This figure is different from the earlier one in that it explicitly represents the patient as a non-sharable resource (resource – constrains represents the pair of NLE's uses and limits). Note that the patient is not needed for all portions of all tests (i.e., the blood work after the blood samples have been drawn). Also note that I have not represented the patient's travel times in any way—this is an important difference from the airport environment described in the next section. The reason for not representing this is that the scale of the problem is such that patient travel time can be fixed beforehand (auxiliaries can assume that the patients will be delivered from their rooms rather than from other auxiliaries). I will go so far as to point out the future implications of this downplay of travel times—this hospital will have to change its organization, i.e. its coordination and scheduling mechanisms, if it has to face significant travel delays with *more than one ancillary*. This could happen if, for example, it is a smaller hospital and cannot afford its own MR scanner or CAT scanner and doctors take to prescribing MR and CAT scans at another hospital. Patients requiring *both* MR and CAT scans now have a significant facilitates effect in their task structures to do both scans nearly consecutively (potentially affecting the coordination structures of *both* hospitals).

The requires – delay relationship says that a certain amount δ of time must pass after executing one method before the second is enabled. I'll repeat the definition I gave in the first chapter here:

$$\text{requires}-\text{delay}(T_a, M, t, d, q, \delta) = \begin{cases} [\mathbf{d}_0(M), 0] & \text{Start}(M) < \text{Finish}(T_a) + \delta \\ [\mathbf{d}_0(M), \mathbf{q}_0(M)] & \text{Start}(M) \geq \text{Finish}(T_a) + \delta \end{cases} \quad (3.15)$$

Examining the hospital's current coordination structure is enlightening because it shows a mismatch between the structure and the current hospital environment (this mismatch having triggered the study in the first place). From this mismatch we can guess at how the environment has changed over time, assuming that the current hospital structure was in fact a good structure when it was originally put into place.[20]

The current hospital structure is described by Ow as follows [Ow *et al.*, 1989]:

> *After receiving a request from a physician, the unit secretary conveys the test request to the secretary of the relevant ancillary department. In turn, the ancillary secretary determined the appropriate time for the test to be run. The unit secretary is notified immediately prior to that scheduled time slot and not sooner. The actual time the patient is scheduled is known only to the ancillary secretary (i.e., it is not relayed back to the requesting unit). Since each ancillary schedules independently (and without knowledge) of all other ancillaries, conflicts arise when a patient is scheduled in overlapping (or nearly overlapping) time slots in different ancillaries. Such conflicts*

[20] Ow calls this the *sympathetic structure*[Ow *et al.*, 1989].

> *must be resolved by the unit secretaries. However, as the unit secretaries are made aware of the scheduled ancillary times only when the request to "deliver" the patient comes from the ancillary, little slack time remains to resolve scheduling conflicts and delays, This can disrupt the care of the patient.*[21]

While this structure seems sorely lacking when compared to the current environment, it may at one time have been a reasonable, low overhead arrangement. It may be that in the past doctors ordered fewer tests on less complex ancillary equipment (there has been quite an explosion in medical technology in the last decade) and (concomitantly) interfering relationships between these technologies. The structure is adapted for a different task environment.

Ow, et al. describe a new set of coordination mechanisms, using computer support, that are better adapted to the hospital's current task environment. Two types of computer agents are defined—a *unit subsystem* that collects, disseminates, and monitors the test requests and resulting schedules for the patients in a unit, and an *ancillary subsystem* that receives test requests from the unit subsystems and schedules them. Two communication protocols rest on this structure:

Primary: The unit subsystems inform the appropriate ancillary about the test request (transmitting part of the task structure). The request includes a soft deadline (the 'flow due date') for the task. The ancillary will schedule the test immediately, using a shared primary performance criteria of not missing the deadline, and secondary local criteria such as minimizing setups, maximizing throughput, etc. All other criteria being equal, the ancillary picks the earliest slot. As soon as the slot is chosen, the scheduled slot (what I will later call a *commitment* in the discussion of GPGP in Chapters 5 and 6) is communicated back to the unit subsystem.

Secondary: As soon as a slot is scheduled for a patient at an ancillary, the slot (commitment) is *broadcast* to all other ancillaries, thus effectively blocking that slot from consideration by the other ancillaries for any tests on that patient (as indicated in the diagram by the resource−constrains relationships.

One thing to note about this new system is that it does not explicitly handle interactions between tests at different ancillaries. The GPGP family of coordination algorithms (Chapters 5 and 6) could be configured to respond to these interactions as well as eliminate broadcast communication.

3.6.3.2 Airport Resource Management

The UMASS ARM (Airport Resource Management [Hildum, 1994]) and Dis-ARM (Distributed ARM [Neiman *et al.*, 1994]) systems solve airport ground service scheduling problems. The function of such systems is to ensure that each airport flight receives required ground servicing (gate assignment, baggage handling, catering, fuel, cleaning, etc.) in time to meet its arrival and departure deadlines [Neiman *et al.*, 1994]:

[21] Additional problems may also ensue. For example, in some cases the *sequence* of multiple tests are important—a wrong sequence can result in patient stay delays as the residual effects of one test may influence the earliest start time of another test [Ow *et al.*, 1989].

The supplying of a resource is usually a multi-step task consisting of setup, travel, and servicing actions. Each resource task is a subtask of the airplane servicing supertask. There is considerable parallelism in the task structure: many tasks can be done simultaneously. However, the choice of certain resource assignments can often constrain the start and end times of other tasks. For example, selection of a specific arrival gate for a plane may limit the choice of servicing vehicles due to transit time from their previous servicing locations and may limit refueling options due to the presence or lack of underground fuel tanks at that gate. For this reason, all resources of a specific type can not be considered interchangeable in the AGSS domain.

The generative model for this environment includes the nominal flight schedule, a model of the errors in this schedule (i.e., late arrival times), and the airport's resources (assuming all planes will need the same set of services). An episode is a 24-hour time period including the initial, nominal flight schedule. Another possible addition to the generative model are failure rates for the servicing resources. Because the nominal flight schedule for the airport is fixed ahead of time, generating an episode mostly concerns variations in arrival time (usually delays) and equipment breakdowns. Each arriving flight spawns a task group to service that flight.

Figure 3.13 shows the objective task structure for a small episode with two gates, two fuel trucks, and two baggage trucks. Gate 2 has an underground fuel tank, and thus enables local fuel delivery without needed a fuel truck. Slightly unusual relationships hold between using a gate and servicing the plane, because the gate must be held for the entire time of servicing (as opposed to a strict sequencing like enables). The subjective information available to any agent is the same, except for the arrival times of task groups (flights) in the future, which are only tentative.

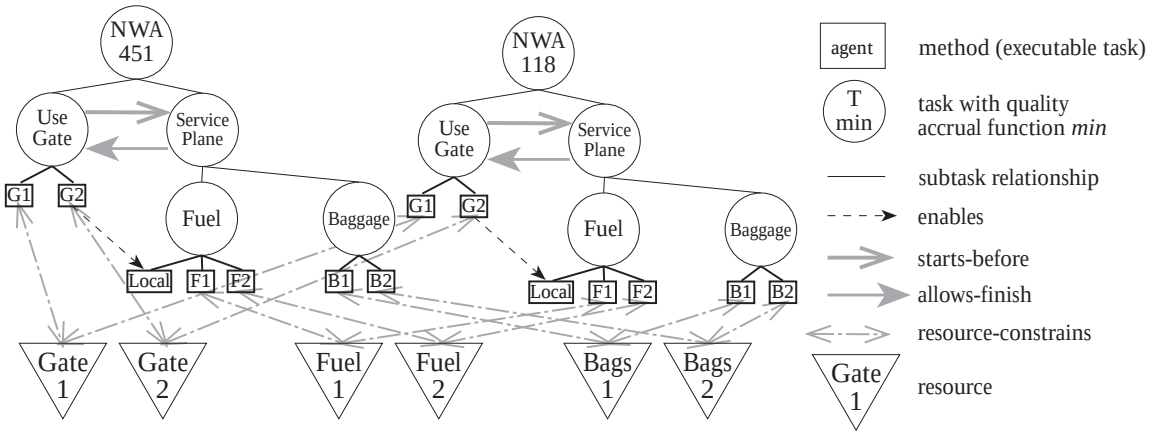

Figure 3.13. Objective task structure for a small airport resource management episode.

Notice the differences between this structure and the hospital structure. Resources are much more important here, and connect much more of the substructures to one another. Much of this comes out of the necessity to represent the effect of equipment travel times that

are not represented in the hospital case (the point is not that 'equipment doesn't travel in the hospital' but that the travel times of the patients can be *standardized*—an important classical coordination technique). Travel times, however, cause every action taken by a resource (such as a fuel truck) to be related to every other action in a complex way, changing the effects at each potential successive action (I'll talk about this again in Section 7.5 below).

This strongly connected structure thus impacts the potential coordination mechanisms used. For example, in ARM, and in real airports, this scheduling is done in a centralized manner for a fixed set of resources (although one might consider resources such as baggage trucks as 'agents', in such a system these agents are slaves that only obey the commands of the central coordinator—similar to the bulldozers in Phoenix[Cohen *et al.*, 1989]).

Such a centralized scheduling process is obviously expensive and complex in terms of computation. The Dis-ARM project [Neiman *et al.*, 1994] looks to provide performance improvements over a centralized system by using multiple schedulers. The subjective view of the problem for each scheduler is considerably simplified by assigning each scheduler exclusive control of certain resources (for example, each scheduler could get one airport concourse of gates and a commensurate number of fuel and baggage trucks). Each scheduler tries to locally schedule the services for planes arriving in its concourse. The problem at each agent then becomes considerably simpler because it has been separated from the similar problems at other agents' concourses, and the agents can solve the problems in parallel as well. The downside is that the totally separate solutions are potentially more wasteful of resources. To combat this, the Dis-ARM system allows agents to lend resources to one another—note however that each lending act will connect initially unrelated task groups through resource relationships and make each agent's scheduling that much harder.

Other solutions are also suggested by the task structure. For example, since the services provided to each plane are already standardized, one could schedule the services at a constant headway (or one keyed to the business pattern, i.e., closer headway during the morning and evening rush times). Thus pre-scheduled and fixed slots would be established, and when a plane arrives (or soon before) it would be assigned to the next available slot (and associated gate). This technique removes the uncertainty in plane arrivals from the system (but is probably not used by airports because they wish for gate assignments to be made as early as possible, both for customers and for automated baggage handling). Another solution would be similar to the Dis-ARM structure; basically the corporate division organizational form—rather than direct negotiation, each scheduler (division) would apply for resources through a centralized 'front office' (which in turn centralizes all the rapidly changing information about the resources). A modification to this structure is a matrix organization where separate agents keep track of each class of resources (i.e., a baggage truck agent, a fuel truck agent, etc.) and then other agents are responsible for servicing the flights, drawing resources from the resource-class pools [Sycara *et al.*, 1991].

3.6.3.3 Internet Information Gathering

Another task environment mentioned briefly in the overview of this thesis was distributed information gathering. The Internet, as well as several popular commercial services, grow by leaps and bounds daily, providing rich and varied sources of information. Quickly and efficiently retrieving this information can be viewed as a set of interrelated tasks in an uncertain

and dynamic environment. The same piece of data can be available via many different methods, and at many different locations; at any point of time only some of those locations are acessable through some subset of methods. It is extremely common nowadays for one to know precisely where a certain answer can be found but for one to have to undertake a time-consuming sequential search to find a currently available resource and access method. Within a single query, multiple agents could search in multiple locations in parallel and coordinate their actions when it was useful. For example, the results of work by one agent may suggest the need for some of the existing agents to gather additional information, or it might suggest the need for a new division of tasks among the agents [Oates *et al.*, 1994].

Generative level information in this task environment includes the frequencies and types of queries (are the agents beholden to individuals, or are they available to the network, which would result in very different usage patterns). It also includes information about the dynamics of the environment: how frequently do connections or links to physical information sources change, how often are the physical sources available, how often does the information on those sources change? How often are new physical sources being added?

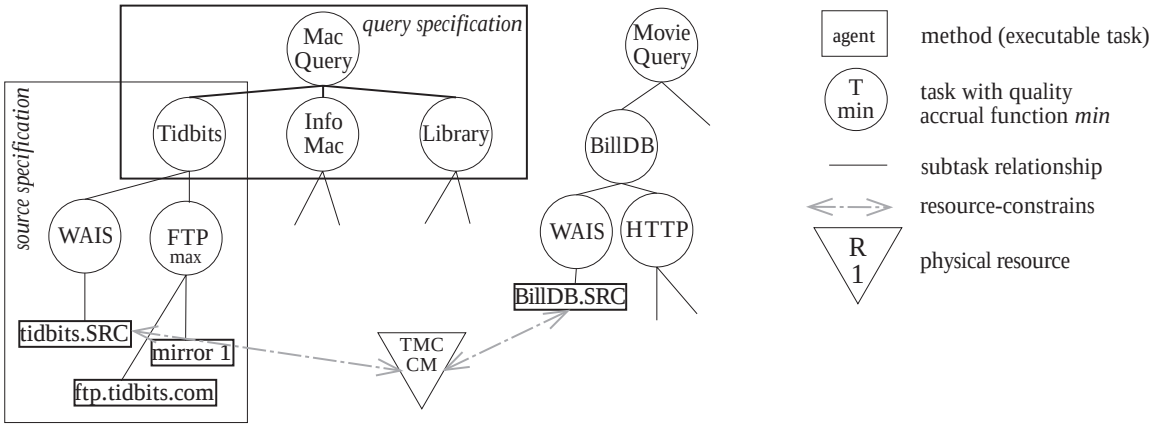

Figure 3.14. High-level, objective task structure for a two independent queries that resolve at one point to a single machine.

Figure 3.14 shows part of an objective level description of an example episode. Here two queries are being made—one about Macintosh information and one about movies. The Mac query might encompass several general sources of Mac information—the TidBits electronic newsletter, Info-Mac and other electronic file archives, Usenet newsgroups, and even standard library article searches. This particular query might be constructed by the interaction of a human user and a persistent Mac query intelligent agent. The task of searching each information source can be broken down into the different possible access methods for that source (e.g., WAIS, FTP, HTTP, a special-purpose agent that can use telnet to access a class of library databases, etc.). Each access method may still imply multiple physical resources (for example TidBits back issues can be found both at the mother site ftp.tidbits.com and at various *mirrors* of that site. Work done retrieving from mirror sites is redundant in the TÆMS sense—but which sites are likely to

be operating at all? How fast will they be? Are they accepting connections?[22] One can imagine the enforcement of user performance criteria on the agents carrying out these tasks. One user might want a (not necessarily complete) answer immediately, another might be willing to wait for the agents to find the 'best' answer they can find. Queries might be continuous 'profiles' for information to be retrieved now and in the future as it becomes available.

Another point to note about Figure 3.14 is the presence of shared resources linking otherwise unrelated queries. In this example the shared physical resource is Thinking Machines' CM5 WAIS server, which stores many example WAIS databases. Other similar examples include popular mirror and archive sites like uunet.uu.net and wuarchive.wustl.edu.

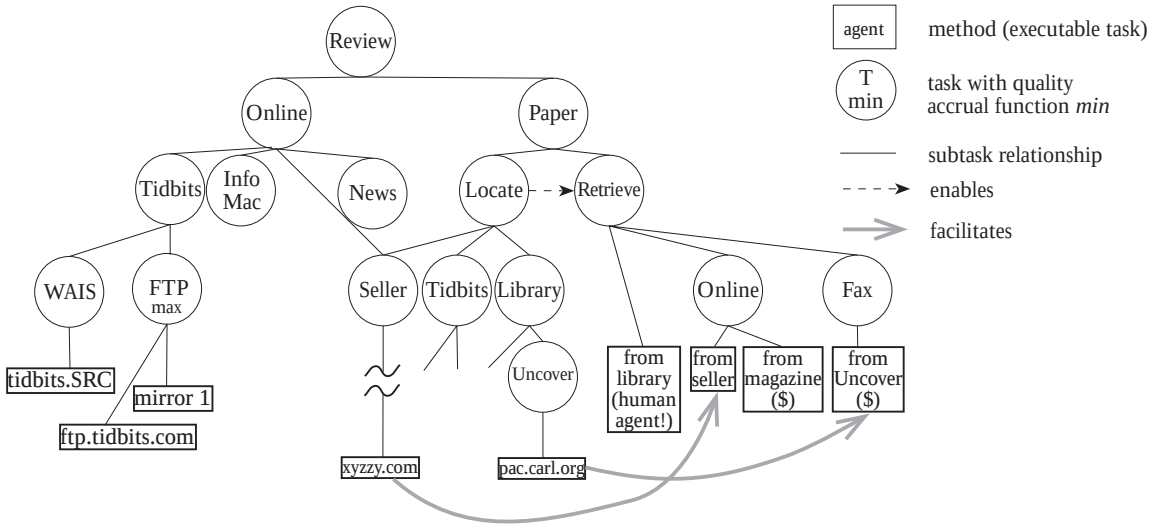

Figure 3.15. Mid-level, objective task structure for a single query for a review of a Macintosh product showing intra-query relationships.

Figure 3.15 shows more detail from a particular Macintosh query episode for a review of some Mac software product. Such reviews can be found both in online forms and in what were originally published paper forms (but which may now be online as well). A user may perhaps view truly published product information as being of higher quality. Online product information might be found in the review portion of the TidBits newsletter, or in collected product information in the Info-Mac archives, or in exchanges about a product in Usenet News. Finding paper reviews of a product can be reduced to locating a citation and then the actual article itself. TidBits contains the table of contents of the popular Macintosh magazines; some magazines are indexed by free periodical indices such as Uncover; sometimes the seller of a product will both have information available on the net and citations of reviews of their product. Once a citation is found, retrieval can be by hand or from online sources (but almost all of these will cost money). Note that some parts of this structure facilitate others, e.g., having

[22] These are all generative level questions.

used Uncover to come up with a citation it is very easy to have the article faxed to you at a price.

The task structure of this environment lends itself to a great deal of parallelism since a lot of work can be done in parallel and the interrelationships are relatively few and fixed. On the other hand, efficient planning for search according to a users preferences (e.g., quick response? best answer?) make the coordination problem more one of intelligent task decomposition.

3.6.3.4 Pilot's Associate

The global coherence problems we would like to address occur in many systems other than the DVMT, such as the Pilot's Associate (PA) system [Smith and Broadwell, 1987], where situations occur that cause potentially complex and dynamically changing coordination relationships to appear between goals that are spread over several agents. Each agent in the Pilot's Associate system has subtasks that other agents must fulfill, and receives tasks from other agents that only it can fulfill.

For example, assume that we are in a tactical situation, so the tactical planner is in control (see Figure 3.16). It has two ordered subtasks: turn on the active sensors (request to situation assessment), and get a detailed route of the plane's movements during the tactical maneuver (request to the mission planner). Turning on active sensors causes a plane to become a transmitter, and thus become easily detected (most of the time the plane uses passive sensors). Since this is dangerous, the situation assessment agent will ask the pilot-vehicle interface (PVI) to ask for pilot confirmation of the use of active sensors. The pilot, upon seeing the request, asks the PVI to plot the escape route of the plane on the screen in case things go wrong. The PVI passes this task to the mission planner.

Meanwhile, the tactical planner has asked the mission planner to produce the detailed route for the tactical maneuver. Which task does the mission planner act on first? From a local view, it may perhaps do the tactical planner request first because the tactical planner tasks are a high priority. But from a global perspective, we see that unless the mission planner plans the escape route, which is needed by the pilot in order to authorize turning on the active sensors, which is needed for the tactical planner to do its job, the whole system performance goal of handling the tactical situation is in jeopardy. Hence the mission planner should do the escape route plan first.

Although I do not describe GPGP in detail until Chapter 5, let me briefly describe this scenario in those terms. There will be an overall deadline on the 'respond to tactical situation' task, which will be inherited by the 'plan tactical route' subtask. The enables relation between 'get pilot confirmation' and 'do it' (pinging the active sensors) could trigger a GPGP mechanism to place an earlier deadline on the 'get pilot confirmation' task when it is communicated to the Pilot Vehicle Interface. The task 'show escape route', based on this, would also have an earlier deadline, and thus 'plan escape route' would have an earlier deadline (earlier than 'plan tactical route') as well. Thus the Mission Planner obtains the necessary information with which to schedule the two tasks. If the time to do both tasks is more than the earlier deadline and less than the later deadline, then the scheduler can only rationally choose to do 'plan escape route' first.

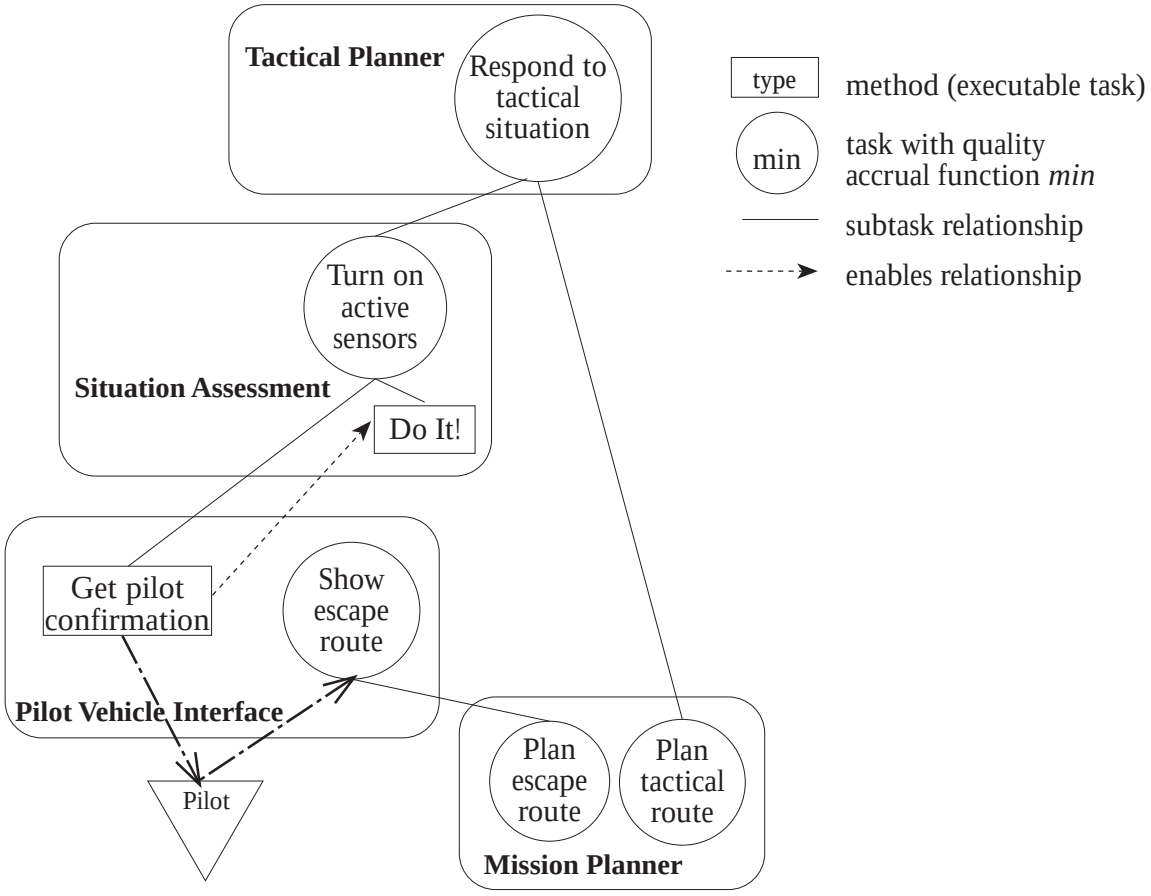

Figure 3.16. Dynamic Situations in Pilot's Associate. All tasks accrue quality with Min (AND).

3.7 Summary

This chapter discussed a characterization of the features and model of the processes in computationally intensive task environments called TÆMS (Task Analysis, Environment Modeling, and Simulation). No characterization currently exists that formally captures the range of features, processes, and especially interrelationships that occur in computationally intensive task environments. In the simplest terms, a TÆMS model of a task environment specifies what actions are available to agents and how those actions relate to one another. This framework is useful not only for the study of coordination (which will be the subject of the next two chapters) and other related CDPS behaviors (for example, our brief discussion of negotiation with respect to Dis-ARM), but also for the study of the planning and scheduling of computation in realistic real-time or parallel task environments [Garvey *et al.*, 1993, Garvey and Lesser, 1993]. The contribution of TÆMS itself is really as a base from which to work—a way to describe an environment carefully, as shown in the examples at the end of the last section. It is also a way to specify the behavior of an agent when you are designing it's control mechanisms, as I will show with GPGP in Chapter 5. Many other things that can be done with TÆMS including analysis and simulation, will be described in the following chapters.

The work described here is of course not without limitations. Some of these are currently

being addressed in follow-up work. TÆMS describes an environment from three different viewpoints, but this dissertation concentrates mostly on the objective viewpoint. It does not deal with the various representations of uncertainty that might be in an agent's subjective view—we are currently working on this aspect. The generative models in this dissertation are constructed from a statistical viewpoint—an episode can be characterized by the values of several random variables. While this is far better than using a set of single-instance examples, it may not be appropriate for dynamic environments, where the task structure itself is changing dramatically over time. More complex models of generation may be needed. For example, generation might be modeled by a set of meta-agents.

Furthermore, especially when modeling human agents and organizations, we only address what might be called "certain specified uncertainty" [Anderton, 1994]—bounded uncertainty about known things. But in real human organizations, often external forces change the very structure of the environment in essentially unpredictable ways. The very factors that influence behavior, the measures of performance, etc., might change out from under the agent with no prior warning (or bounding of the uncertainty surrounding these matters). We'll discuss this more in Chapter 2. Another problem is if any such general framework could achieve an external validity that doesn't require constant re-verification in a new domain. While this seems possible for computational environments, it doesn't seem possible for human systems.

The TÆMS representation of task environments is abstract, with the smallest units corresponding to schedulable actions and their relationships (what that means will depend on the environment being modeled—anything from an indivisible set of real-time tasks to an abstract organizational unit action). The TÆMS representation is *not* intended as a schedule or plan representation, although it provides much of the information that would go into such uses. TÆMS has many dynamic features including:

- dynamic stochastic arrival of whole or partial task structures,
- dynamic changes to interrelated task durations and qualities caused by the actions of other agents and stochastic environmental processes
- dynamic changes to the agents' local views as time progresses and caused by local and non-local actions

The concept of agency in TÆMS is based on simple notions of *execution*, *communication*, and *information gathering* (collectively, *actions*). An agent is a locus of belief (state) and action. By separating the notion of agency from the model of task environments, we did not have to subscribe to particular agent architectures (which one would assume will be adapted to the task environment at hand). TÆMS exists happily alongside blackboard systems, Shoham's agent oriented programs, or other agent architectures. Such a conception is unique among computational approaches. We'll talk about blackboard-style (DVMT) agents in the next section, and you'll see our use of an architecture similar to Shoham's agent oriented programs in Chapter 5 (GPGP).

The framework is general in that it is based on abstract features, applicable to many domains. It is not a model of a specific domain, such as sensor interpretation or fire fighting. We showed models of several different environments and episodes in this chapter; we'll expand on the simple distributed sensor network model in the next chapter.

A unique feature of the TÆMS framework is the direct representation of interrelationships between tasks beyond resource and precedence constraints. Coordination relationships are crucial to design and analysis of coordination mechanisms, as you will see in chapters 5 and 6. TÆMS allows the formal functional specification of arbitrary task interrelationships; Section 3.4.3.1 described a useful basic subset.

Finally, this chapter demonstrated, as promised, that TÆMS models are built in three layers—*objective*, *subjective*, and *generative*. This allows us to capture the single problem instances that have appeared in many previous non-formal approaches to CDPS systems (i.e. the DVMT, hospital scheduling, and other examples in the final section). We can also use it to look at wider classes of problems with stochastically varying features from a statistical perspective—and that is exactly the subject of the next chapter.

CHAPTER 4

DESIGN AND ANALYSIS OF COORDINATION ALGORITHMS: A TASK ENVIRONMENT MODEL FOR A SIMPLE DSN ENVIRONMENT

Now I know a problem that physics won't resolve
. . . I know a color that acid won't dissolve
. . . a place that's just too far I think for us to go

Now is it wise to have a hope so strong
for real-time answers to real-world wrongs?
— Game Theory, Distortion:"Nine Lives to Rigel Five", 1984

In the previous three chapters I described the TÆMS framework and several example models, including one of a simple Distributed Sensor Network (DSN) environment. This chapter takes this model and uses it to analyze the DSN problem in some depth. Previous approaches to analyzing organizations in distributed sensor networks have either not focused on the effectiveness of the organization [Davis and Smith, 1983, Pavlin, 1983], or have only analyzed organizational effectiveness in particular, single-instance examples [Durfee *et al.*, 1987]. Others work has focussed on analysis in general situations, but is extremely abstract [Malone, 1987]. Sen has done an analysis similar in spirit to that presented here, but in a different domain (meeting scheduling for two agents)[Sen and Durfee, 1994]. I will develop and test mathematical expressions for the important TÆMS characteristics of objective episodes in the DSN environment, and use these expressions to develop and analyze coordination algorithms for agents in this environment. By the end of this chapter I will have developed expressions for the primary performance criteria in this environment—termination time—for two different coordination algorithms. I validate these mathematical models by using simulations.

A dynamic organization is one in which the responsibilities of agents can be reassigned based on a developing view of the problem at hand. Agents that have a dynamic organization have the option of meta-level communication—communicating about the current state of problem solving as opposed to communicating about solving the problem itself. Due to the uncertainties explicitly represented in the TÆMS task environment model, there may not be a clear performance tradeoff between static and dynamic organizational structures in every environment.

This chapter will serve as a concrete example of the impact of environmental differences on coordination algorithms. Although I will discuss three different algorithms and implement two of them, no one algorithm will be the 'best', even statistically, for all DSN environments. If variance in an environment is large enough, it will behoove the agents to use meta-level communication to build a dynamic organization that, as Stinchcombe puts it, expands toward the earliest available *information that resolves uncertainties* in the current environment, allowing the agents to then create an efficient organization for the situation.

4.1 Task Environment Simulation

In the next section, we will test the model we are developing against a simulation of DSN problem solving. Each simulated DSN episode will take place on a grid where the concepts of length and size correspond directly to physical distances. For example, Figure 4.1 illustrates several simple organizations imposed on such a grid in our simulation.

In the simulation we assume that each vehicle is sensed at discrete integer locations (as in the DVMT), randomly entering on one edge and leaving on any other edge. Between these points the vehicle travels along a *track* moving either horizontally, vertically, or diagonally each time unit using a simple DDA line-drawing algorithm (see Figure 4.6). In an 18×18 grid, the (empirical) average length of a track is 14 units—the actual length of any one track will range from 2 to 19 units and is not distributed normally. Given the organization (r, o, and A, and the geometry), we can calculate what locations are seen by the sensors of each agent. This information can then be used along with the locations traveled by each vehicle to determine what part of each task group is initially available to each agent. In general, no agent will see all the parts of a complete task group (track). Section 4.5 will detail what the structure of each task group is for the DSN simulation.

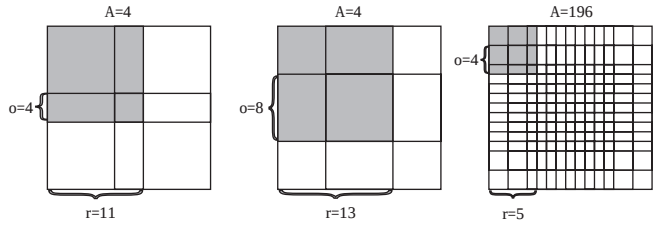

Figure 4.1. Examples of DSN organizations on an 18×18 grid

Our TÆMS generative level model of this process was discussed back in Section 3.3. The properties of the episodes in a DSN environment are summarized by the tuple $\mathcal{D} =< A, \eta, r, o, \mathcal{T}(l) >$ where A specifies the variable number of agents, η the expected number of task groups, r and o specify the variable structural portion of the organization by the *range* of each agent and the *overlap* between agents, and $\mathcal{T}(l)$ specifies a template for the structure of each task group. A particular episode in this environment can be described by the tuple $\mathbf{D} =< \mathcal{T}_1(l_1), \ldots, \mathcal{T}_n(l_n) >$, where n is a random variable generated from a Poisson distribution with location parameter (central tendency) of η. The l_i are generated by the process explained above.

This specification is applicable to other interesting environments. For example: each task group may comprise several different types of subtasks; each agent may only respond to a certain type of subtask (its 'range'); multiple agents may respond to the same types ('overlap'). In general, the parameters of 'range' and 'overlap' can be multidimensional.

4.2 Expected Number of Sensor Subtasks

When a system of agents is confronted by a particular DSN problem-solving episode, each agent will *sense*, or *'see'* [1] a certain amount of low level data associated with the tracks of vehicles that moved through the agent's sensed area. In order to analyze the performance of a particular organization, we will want to know (*a priori*) what proportion of each task group each agent is likely to process. There will be some upper limit on this proportion (related to the agent's range r), and sometimes the agent will process less than this upper limit. Especially in static organizational structures where tasks are not exchanged, the termination of the system as a whole can be tied to the completion of all tasks at the most heavily loaded agent. Normally, we would use the *average* part of a task group to be seen, but since the focus of our analysis is the termination of problem solving, we need to examine the *expected maximum* portion of a task group to be seen. This section will develop an equation for the expected maximum workload at an agent by counting the expected number of low-level sensor subtasks (each individually associated with a sensed vehicle location) that the maximally loaded agent will have.

The amount of a single task group seen by an agent (which is the same as the number of sensor subtasks in the DSN example) can be viewed as a random variable S with a probability density function and corresponding cumulative distribution function. In the DSN environment, S is discrete, and its probability function (determined empirically[2]) is heavily weighted toward r (the maximum). To simplify the analysis, instead of letting S correspond to the number of subtasks in a single task group seen by an agent, let's create a new random variable $\mathbf{S}$ that we have equal 1 if the agent sees the maximum amount, and 0 otherwise. Now $\mathbf{S}$ has a Bernoulli (coin-tossing) distribution with parameter p corresponding to the chance of an agent seeing the maximum amount r of a task group.[3] Let's assume we know that $N \leq n$ is the number of task groups at the maximally loaded agent, and that on average $a \leq A$ agents see a single task group (we'll remove these assumptions later). The number of times an agent sees the maximum out of N task groups (N coin flips) then has a binomial distribution ($b_{N,p}(s)$). We need to know, given that a agents each flip N coins, what the distribution is of the *maximum* number of 'heads' any agent sees—this is called the binomial max order statistic, $g_{a,N,p}(s)$:[4]

$$\begin{aligned} b_{N,p}(s) &= \binom{N}{s} p^s (1-p)^{N-s} && [\Pr[\mathbf{S} = s]] \\ B_{N,p}(s) &= \textstyle\sum_{x=0}^{s} b_{N,p}(x) && [\Pr[\mathbf{S} \leq s]] \\ g_{a,N,p}(s) &= B_{N,p}(s)^a - B_{N,p}(s-1)^a && [\Pr[\hat{\mathbf{S}} = s]] \end{aligned}$$

The random variable $\mathbf{S}$ refers to *any* agent, and the new random variable $\hat{\mathbf{S}}$ refers to the *maximally loaded agent*. Its probability function $g(s)$ has a much steeper shape and larger expected value than the binomial $b(s)$ (see Figure 4.2). In the DSN example we have $p = 0.5$[5] and the amount of a task group seen when an agent does not see the maximum amount averages $r/2$. In effect, we are modeling the sensor reception process by the simplification that when

[1] As is common with the DVMT, we will use "sensed" and "seen" interchangeably in this chapter.

[2] I built a simulation and counted it in over 500 random episodes.

[3] It is entirely possible to complete the analysis using the empirical, discrete distribution function, but it adds little to the effectiveness of the model and needlessly complicates the discussion.

[4] A more detailed derivation of this result is in our tech report.

[5] Empirically determined through simulation as previously mentioned.

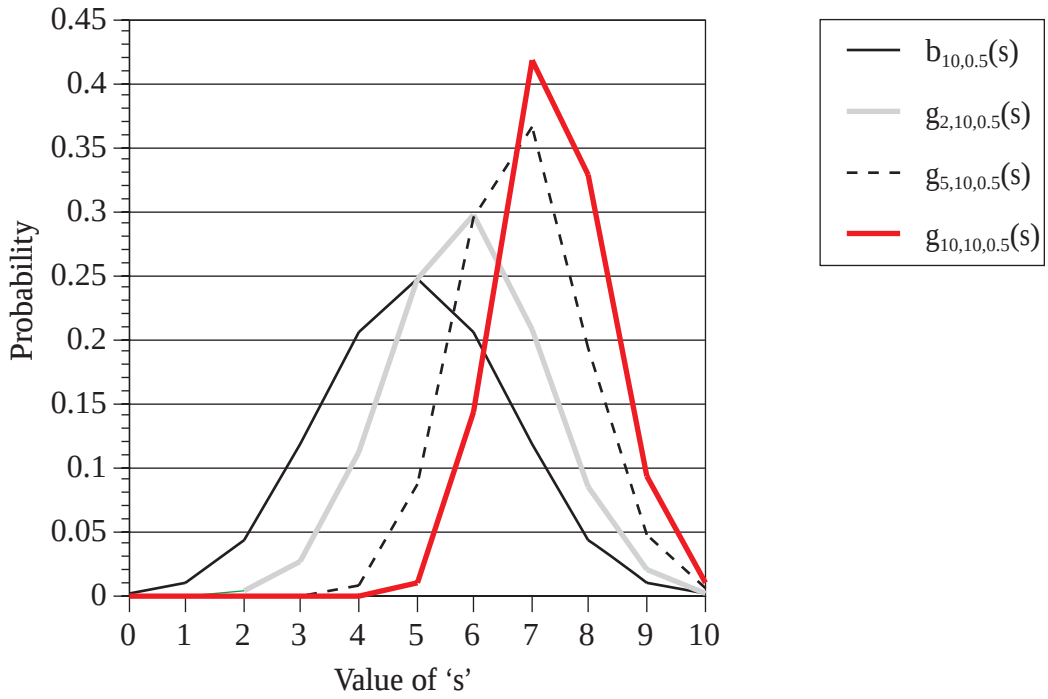

Figure 4.2. A comparison of the probability distributions of the outcome of 'heads' in the act of flipping a coin 10 times (the binomial $b_{10,0.5}(s)$) compared to the outcome of having 2, 5, or 10 agents flip 10 coins and taking the number of heads of the agent who flipped the most heads (the max order statistics $g_{2,10,0.5}(s)$ through $g_{10,10,0.5}(s)$).

the coin is heads we see the maximum r sensed data points, and when the coin is tails we see only $r/2$. Now we can convert back from the easy to analyze random variable $\hat{\mathbf{S}}$ to the variable we are really interested in, which I will call $\hat{S}$, by the equation $\hat{S} = (r\hat{\mathbf{S}} + (r/2)(N - \hat{\mathbf{S}}))$. The new random variable $\hat{S}$ models how many sensed data points are seen by the agent who sees the most complete tracks.

Remember that the definition of expected value is the sum of the product of the probability of a value and the value itself across all possible outcomes. In the case of $\hat{S}$, the probability of any particular value is given by the appropriate max order statistic g that some agent sees s 'heads', and the number of sensed data points is given by the function $(r\hat{\mathbf{S}} + (r/2)(N - \hat{\mathbf{S}}))$ (again, when there were s 'heads'). Therefore, the expected heaviest load seen by any agent when a agents see N task groups with a probability p of seeing r and probability $1-p$ of seeing $r/2$ is:

$$E[\hat{S}|N,a] = \sum_{s=0}^{N} g_{a,N,p}(s)(rs + \frac{r}{2}(N-s)) \tag{4.1}$$

To reiterate: out of N task groups the maximally loaded agent sees the maximum r some $\hat{\mathbf{S}}$ (a random variable) times, and the other $(N - \hat{\mathbf{S}})$ times it sees only $r/2$. Eq. 4.1 shows the expected value of a new random variable $\hat{S}$ that indicates the number of sensor subtasks at the *maximally* loaded agent when there are N task groups in an episode. N is itself a random variable, we'll look at its distribution in the next section.

Figure 4.3 shows the heaviest load actually observed and averaged over 1000 runs, plotted against the expected value, for n tracks and all square DSN organizations $[2 \leq r \leq 10, 0 \leq o \leq r, 1 \leq \sqrt{A} \leq 10, 1 \leq N \leq 10]$ $(R^2 = 0.98)$[6]. The colors of the points refer to the value of r; lighter grey corresponds to larger values of r.

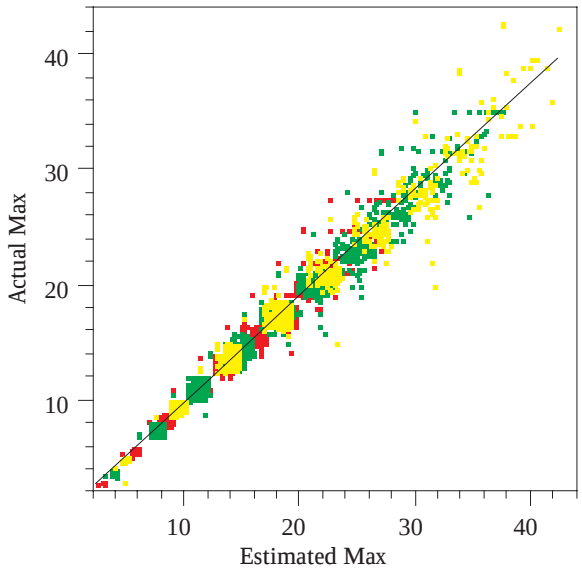

Figure 4.3. Actual versus predicted heaviest load $\hat{S}_N$ for various values of A, r, o, and N

4.2.1 Distribution of the Binomial Max Order Statistic

To expand on the formulae in Section 4.2, the amount of a single task group seen by an agent can be viewed as a random variable S with probability density function $f(s)$ and corresponding cumulative distribution function $F(s)$. Let a be the number of agents that initially see part of a single task group. By elementary statistics, the density of the max order statistic $S_{max} = \max(S_1, S_2, \ldots, S_a)$ is $g_a(s) = aF(s)^{a-1}f(s)$. The expected heaviest load then is $\int_0^r s g_a(s) ds$. In the DSN environment, S is discrete, and its probability function (determined empirically) is heavily weighted toward r (the maximum). To simplify the analysis, instead of letting S correspond to the number of subtasks in a single task group seen by an agent, we again introduce a new random variable $\mathbf{S}$ equal 1 if the agent sees the maximum amount, and 0 otherwise. Now $\mathbf{S}$ has a Bernoulli (coin-tossing) distribution with parameter p corresponding to the chance of an agent seeing the maximum amount r of a task group. $\mathbf{S}$ then corresponds to the number of times an agent sees the maximum r if the agent sees N task groups (tracks). $\mathbf{S}$ has a binomial distribution with parameters p and N. Now if a agents see N task groups each, what is the distribution of $\hat{\mathbf{S}} = \max(\mathbf{S}_1, \mathbf{S}_2, \ldots, \mathbf{S}_a)$? This is the max

[6] R^2 is the squared correlation coefficient, a measure of goodness of fit. It may be interpreted as the proportion of total variability in the observed data that is explained by the model.

order statistic for the binomial distribution, and because it is discrete, can be easily derived. The probability function and cumulative distribution function for the binomial distribution are:

$$\begin{aligned} b_{N,p}(s) &= \binom{N}{s} p^s (1-p)^{N-s} \quad [\Pr[\mathbf{S} = s]] \\ B_{N,p}(s) &= \sum_{x=0}^{s} b_{N,p}(x) \quad [\Pr[\mathbf{S} \leq s]] \end{aligned}$$

If we assume each track is independent of the others, we can derive the cumulative distribution function:

$$\begin{aligned} \Pr[\hat{\mathbf{S}} \leq s] &= \Pr[\max(\mathbf{S}_1, \mathbf{S}_2, \ldots, \mathbf{S}_a) \leq s] \\ &= \Pr[\mathbf{S}_1 \leq s] \Pr[\mathbf{S}_2 \leq s] \cdots \Pr[\mathbf{S}_a \leq s] \\ &= B_{N,p}(s)^a \end{aligned}$$

The probability function $g_{a,N,p}(s) = \Pr[\hat{\mathbf{S}} = s]$ is then:

$$g_{a,N,p}(s) = B_{N,p}(s)^a - B_{N,p}(s-1)^a$$

4.3 Expected Number of Task Groups

Given the maximum number of task groups seen by an agent (N), we can calculate the expected heaviest agent load using Equation 4.1. But this begs the question of what is the maximum number of task groups an individual agent will see, given the actual (n) or expected number (η) that the entire system will see. The solution is similar—each agent either sees or does not see each of the n task groups, another binomial process. Let N_i be the number of task groups sensed by agent i, with a binomial distribution of parameters n and q. If a is again the number of agents that see a single task group and A the total number of agents, then $q = a/A$, the probability that each agent will see a particular track (we'll give an equation for a next). By the same derivation as in the last section, the max order statistic $\hat{N}$ has the probability function $g_{A,n,a/A}(s)$, and expected value:

$$E[\hat{N}|n,a] = \sum_{s=0}^{n} s g_{A,n,a/A}(s) \tag{4.2}$$

Figure 4.4 shows the actual mean value of the maximum number of tracks seen by an agent over 1000 runs of the DSN simulation, versus the predicted value, for n tracks and square DSN organizations $[2 \leq r \leq 10, 1 \leq \sqrt{A} \leq 10, 1 \leq n \leq 10]$ without any overlap ($R^2 = 0.96$)

4.4 Expected Number of Agents

The only remaining term we need to analyze before deriving an expression for system performance is a, the expected number of agents that will see a single task group. In general, a will depend on the total number of agents A and the organization (r and o). When there is only one agent, it will see every task group ($a = 1$). When the agents overlap completely, every agent sees every task group ($[o = r] \rightarrow [a = A]$). When the agents in a square environment

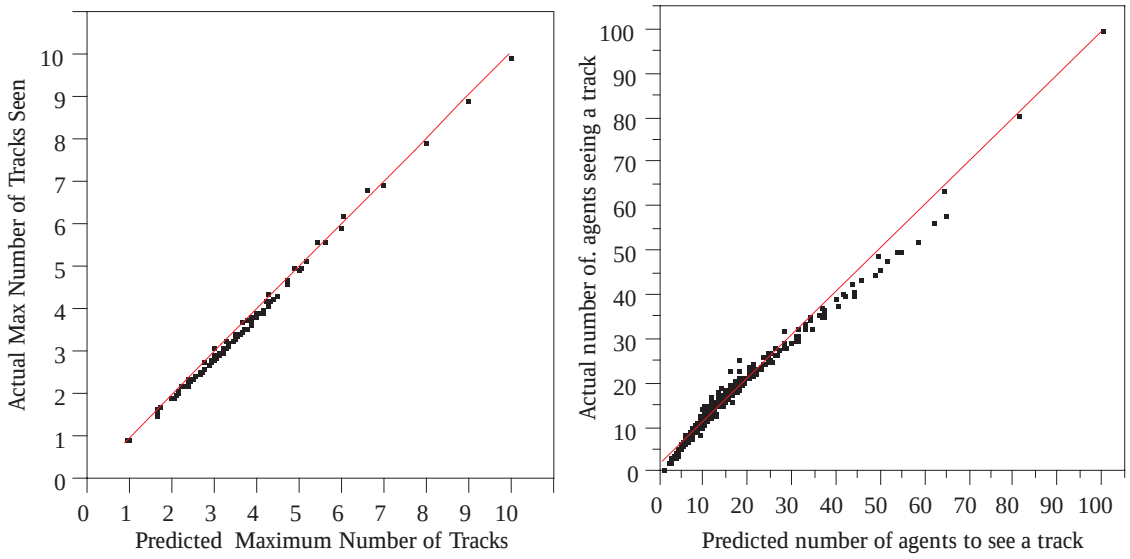

Figure 4.4. On the left, actual versus predicted maximum number of task groups (tracks) seen by any one agent for various r, A, and n. On the right, actual versus predicted average number of agents seeing a single task group (track) for various r, o, and A.

do not overlap, a is approximately $\sqrt{A}$. The relationship follows the ratio of the area solely covered by an agent plus the area of the overlapping section, to the total area covered alone:

$$a = A^{(\frac{r^2+o^2}{2r^2})} \tag{4.3}$$

Note that a is not a random variable, it is just derived directly from environmental parameters. Figure 4.4 shows a regression of the actual average value of a over 1000 runs verses the predicted value for all 630 DSN organizations $[2 \leq r \leq 10, 0 \leq o \leq r, 1 \leq \sqrt{A} \leq 10]$ ($R^2 = 0.98$).

4.5 Work Involved in a Task Structure

So far we have derived models for the number of low-level sensor subtasks seen by the most heavily loaded agent ($\hat{S}$), the number of task groups seen by that agent ($\hat{N}$), and the average number of agents to see a task group (a). Now we turn to the question of just how much work an agent that sees S sensed data points and N task groups has to do in order to solve the problem at hand. Our model of the performance of the system as a whole is based on the structure of the tasks involved. We will use TÆMS to describe this structure.

Let me briefly restate the basics of TÆMS from the previous chapter. The *objective* level describes the essential structure of a particular problem-solving situation or instance over time. It focuses on how task interrelationships dynamically affect the *quality* and *duration* of each task. In this chapter we will concentrate only on duration as a performance metric. An individual task that has no subtasks is called an executable method M and is the smallest schedulable chunk of work. The quality and duration of an agent's performance on an individual task is a function of the timing and choice of agent actions ('local effects'), and possibly previous task

executions ('non-local effects'). The basic purpose of the objective model is to formally specify how the execution and timing of tasks affect quality and duration.

Elsewhere in this dissertation we consider the case of *facilitation*, a non-local effect where the availability of a result from one task alters the quality and duration of another task, but the non-communication of a result has no effect [Decker and Lesser, 1993a]. This chapter considers a different relationship, *precedence*. If task A precedes task B, then the maximum quality $\mathbf{q}(B,t) = 0$ until A is completed and the result is available, when the maximum quality will change to the initial maximum quality $\mathbf{q}(B,t) = \mathbf{q}_0(B)$.

4.5.1 Execution Model

For this chapter we use an extremely simple model of execution (as described earlier in Section 3.5.3.1). Agents can perform three actions: method execution, communication, and information gathering. The control component of an agent determines the next action an agent will perform based on the agent's current set of beliefs. A method execution action, of method M, that is begun at time t will conclude at time $t + \mathbf{d}(M,t)$. An information gathering action has duration $\mathbf{d}_0(I)$ and updates the agent's set of beliefs with any new information in the environment, for example, the arrival of data at the start of an episode, or communications from other agents. A communication action has duration $\mathbf{d}_0(C)$ and, after a communication delay, makes information (such method execution results) available to other agents. The agent on the receiving side must perform an information gathering action before the communication can affect its local beliefs.

4.5.2 Simple Objective DSN Model

Recall from Chapter 3 that the summary of a DSN problem-solving environment was the tuple $\mathcal{D} =< A, \eta, r, o, \mathcal{T} >$. A particular episode in this environment can be described by the tuple $D =< A, r, o, \mathcal{T}_1, \ldots, \mathcal{T}_n >$ where n is a random variable drawn from an unknown distribution with location parameter (central tendency) of η. Note that we so far make almost no assumptions about this distribution; its characteristics will differ for different environments. For example, in the description of our DSN simulation early in Section 4.1 we noted the physical process by which vehicle tracks were generated and that the length of the tracks was not normally distributed.

Each task group $\mathcal{T}i$ is associated with a track of length l_i and has the same basic objective structure, based on the DVMT (repeated here from Chapter 3):

- l_i Vehicle Location Methods (VLM's) that represent processing raw signal data at a single location to a single vehicle location hypothesis.
- $l_i - 1$ Vehicle Tracking Methods (VTM's) that represent short tracks connecting the results of the VLM at time t with the results of the VLM at time $t + 1$.
- 1 Vehicle Track Completion Method (VCM) that represents merging all the VTM's together into a complete vehicle track hypothesis.

Non-local enables effects exist between each method at one level and the appropriate method at the next level as shown in Figure 3.2—two VLMs enable each VTM, and all VTM'S enable the lone VCM.

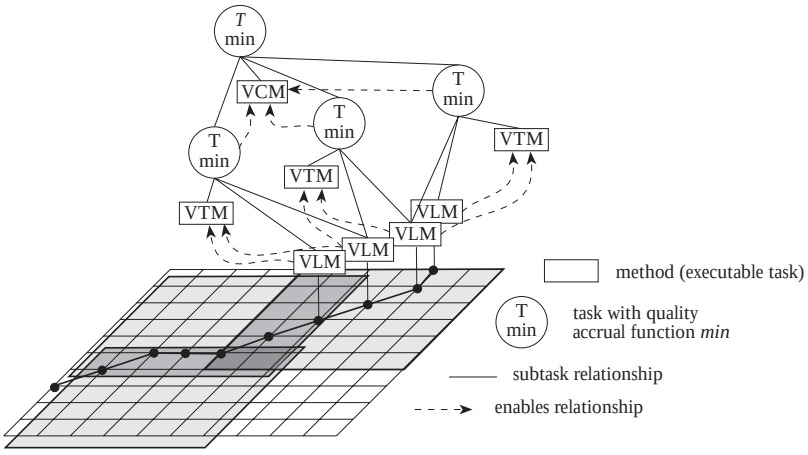

Figure 4.5. Objective task structure associated with a single vehicle track.

If we assume that each VLM has initial duration $\mathbf{d_0}(\text{VLM})$ and each VTM has the initial duration $\mathbf{d_0}(\text{VTM})$, then we can see from the task structure that for each task group the total execution time taken by a single processor agent will be:

$$l_i\mathbf{d_0}(\text{VLM}) + (l_i - 1)\mathbf{d_0}(\text{VTM}) + \mathbf{d_0}(\text{VCM}) \tag{4.4}$$

This task structure is a simplification of the real DVMT task structure. For example, there is no sensor noise (which will cause *facilitation* relationships between tasks, and there is no confusion caused by 'ghost tracks'. Adding these features to the task structure can cause interesting phenomena as was previously discussed in Section 3.5.4. That section explained how to add these features to this basic task structure using TÆMS.

4.6 Model Summary

This section has described our basic model of the DSN environment, beginning with the basic parameters $\mathcal{D} =< A, \eta, r, o, \mathcal{T} >$. A particular episode in this environment can be indicated as $D =< A, r, o, \mathcal{T}_1, \ldots, \mathcal{T}_n >$, where each of the n task groups has the structure mentioned above. This section also developed expressions for $\hat{S}$, the number of low-level sensor subtasks (i.e., VLM's) at the most heavily loaded agent, the number of task groups $\hat{N}$ at the most heavily loaded agent, and the average number of agents a that see a single task group. Another way to look at these results is that we have derived the probability distributions of these variables, i.e., if the system of agents as a whole sees n total task groups, then the distributions of $\hat{N}$ and $\hat{S}$ are:

$$\Pr[\hat{N} = N|n] = g_{A,n,\frac{a}{A}}(N) \tag{4.5}$$

$$\Pr[\hat{\mathbf{S}} = s|\hat{N} = N] = g_{a,N,0.5}(s) \tag{4.6}$$

$$\hat{S} = (r\hat{\mathbf{S}} + (r/2)(N - \hat{\mathbf{S}})) \tag{4.7}$$

Finally Eq. 4.4 derived the duration, or amount of processing work, involved in a single task group from its structure.

In the next section we will combine these results with simple local scheduling and coordination algorithms appropriate for static and dynamic organizations. This will allow us to describe the performance of a system of agents following one of these organizational algorithms in a particular episode or environment.

4.7 Static vs. Dynamic Organizational Structures: Reorganizing to Balance System Load

Now we have the necessary background to analyze static and dynamic organizational structures. We define a static structure as one that does not change during a problem-solving episode. A dynamic structure will change during an episode—specifically, the agents may redistribute their processing in the overlapping regions of their sensor ranges. We have equations for the maximum expected number of subtasks at an agent given the number of task groups seen $\hat{S}$, the maximum expected number of task groups seen given the total number $\hat{N}$, and the predicted number of agents sensing part of a task group a. The key to static structures is to divide up the overlap area *a priori* (rather than to allow agents to do redundant work in the overlap area [Durfee *et al.*, 1987]). The key to dynamic organizational structures is to shift the dividing line in the overlap area, or to transfer tasks, so that all the agents' resources are used efficiently.

We will repeat the assumptions we discussed at the start of Section 4.1 on page 86: the agents are homogeneous (have the same capabilities with respect to receiving data, communicating, and processing tasks), the agents are cooperative (interested in maximizing the system performance over maximizing their individual performance), the data for each agent in an episode is available simultaneously to each agent as specified by their initial organization, and there are only structural (precedence) constraints within the subtasks of each task group. Note that each agent only gets ("sees") the data in its sensor region.

4.7.1 Analyzing Static Organizations

In a static organization, agents divide the overlapping areas of their ranges as evenly as possible. The result is a new area of responsibility $r' = r - \frac{o}{2}$ for each agent with no overlap (see Figure 4.6).[7] Given the task structure as described in Section 4.5 and shown in Figure 3.2, and any raw data or communicated task results provided by information gathering actions, the agent can at any time build a list of currently executable methods (under the set of precedence constraints). Also, at any time an agent can build a list of methods that need to be executed, but cannot be because their precedence constraints have not yet been met. The communication action in this algorithm is a broadcast of the highest level results of all the task groups an agent has worked on. In DVMT terms, we are joining the partial track that has been constructed at one agent with the partial track constructed at a second agent, to produce a single, longer track. Each agent follows the same control algorithm (remember, all the raw data is available at the start) and terminates when all task groups are completed (either locally or by reception of the result from another agent):

[7] The reason for overlap will be apparent in dynamic structures—multiple agents can work in an overlapping area without paying any cost for communicating raw data between them. Overlap can also provide redundancy in case of agent failure.

```
(Repeat
        Do Information-Gathering-Action
        (Repeat
                Let E = [get set of currently executable methods]
                (For method In E
                        Do Method-Execution-Action(method))
        Until (null E))
        Do Communication-Action(broadcast highest-level results)
        Let W = [get set of methods still waiting on precedence constraints]
Until (null W))
```

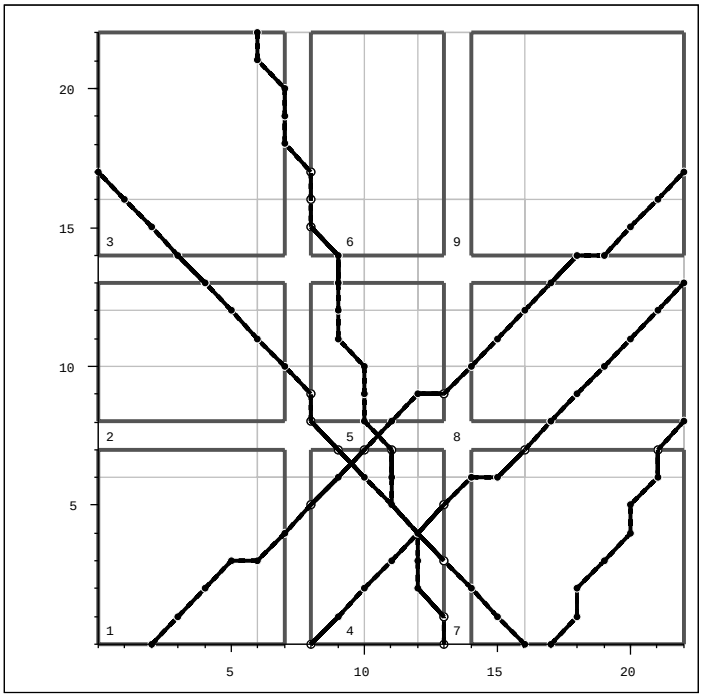

Figure 4.6. Example of a 3x3 organization, $r = 11$, $o = 5$, with 5 tracks. The thick dark grey boxes outline the default static organization, where there is no overlap.

First, let us analyze this algorithm assuming that only *one* task group (vehicle track) is present. In the environment $\mathcal{D} =< A, \eta, r, o, \mathcal{T} >$, if we let S' represent the largest amount of low-level data in *one* task group seen by any agent, and a the total number of agents that see the task group (from the agents' organization, a is defined in Equation 4.3), then the amount of time it will take that agent to construct a complete solution is equal to the amount of time it will take for the initial information gathering action $(\mathbf{d}_0(I))$ plus the amount of time to do all the local work $(S'\mathbf{d}_0(\text{VLM}) + (S' - 1)\mathbf{d}_0(\text{VTM}))$, communicate that work $(\mathbf{d}_0(C))$, get the other agents' results $(\mathbf{d}_0(I))$, plus the amount of time to combine results from the other $a - 1$ agents $((a - 1)\mathbf{d}_0(\text{VTM}))$, plus time to produce the final complete task group result $(\mathbf{d}_0(\text{VCM}))$, plus communicate that result to everyone $(\mathbf{d}_0(C))$. For simplicity we will assume

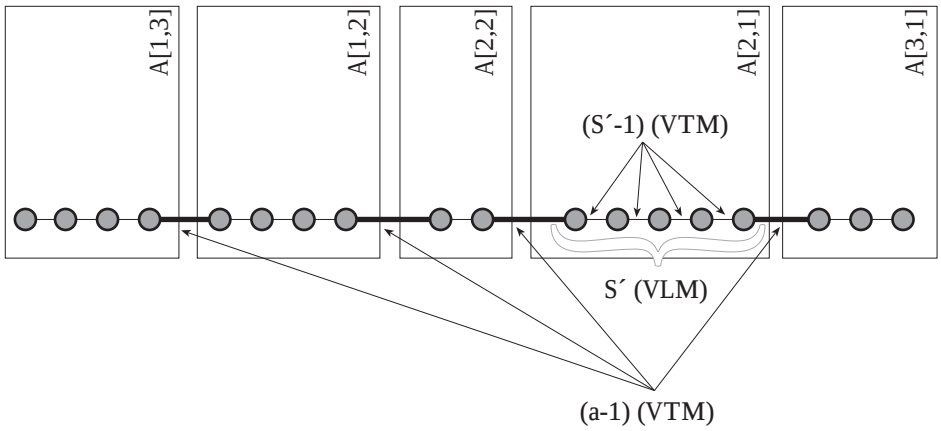

Figure 4.7. Detail from the previous figure of the processing that takes place on the track running from (1,17) to (16,1)

that $\mathbf{d}_0(I)$ and $\mathbf{d}_0(C)$ are constant and do not depend on the amount of data. Note also that since this agent is the most heavily loaded, by definition all other agents will finish their work and have communicated it by the time this agent finishes its local work.

For example, Figure 4.7 shows the meaning of this formula with respect to the single track in Figure 4.6 that runs from point (1,17) to point (16,1). In this example, $S' = 5$, the number of sensed data points at agent [2,1]. Agent [2,1] will have to execute 5 VLM's which will take time $5\mathbf{d}_0(\text{VLM})$. To join the vehicle locations into a local partial track will take $S' - 1$ VTM's, in this case, 4. In this example, $a = 5$, the number of agents that see this track. Therefore the work needed to connect the 5 partial tracks is in this case $a - 1 = 4$ more VTM's.

In the general case, if the system sees $n = \eta$ total task groups, then the expected amount of low-level sensor data (size of the initial data set) at the maximally loaded agent can be derived from the marginal expected value for $\hat{S}$ given the joint distribution of $\hat{S}$ (Eqns. 4.6, 4.7) and $\hat{N}$ (Eq. 4.5):

$$E[\hat{S}] = \sum_{N=0}^{n} \sum_{s=0}^{N} g_{A,n,\frac{a}{A}}(N) g_{a,N,p}(s)(rs + \frac{r}{2}(N - s)) \tag{4.8}$$

Similar to the single task group case, the total time until termination for an agent receiving an initial data set of size $\hat{S}$ is the time to do local work, combine results from other agents, and build the completed results, plus two communication and information gathering actions:

$$\hat{S}\mathbf{d}_0(\text{VLM}) + (\hat{S} - \hat{N})\mathbf{d}_0(\text{VTM}) + (a-1)\hat{N}\mathbf{d}_0(\text{VTM}) + \hat{N}\mathbf{d}_0(\text{VCM}) + 2\mathbf{d}_0(I) + 2\mathbf{d}_0(C) \tag{4.9}$$

We can use Eq. 4.9 as a predictor by combining it with the probabilities for the values of $\hat{S}$ and $\hat{N}$ given in Eqns. 4.6, 4.7, and 4.5 (this is very similar to the treatment in Eq. 4.8).

We tested these predictions of Equation 4.9 versus the mean termination time of our DSN simulation over 10 repetitions in each of 43 randomly chosen environments from the design space $[2 \leq r \leq 10, 0 \leq o \leq r, 1 \leq \sqrt{A} \leq 5, 1 \leq N \leq 10]$[8]. The durations of all tasks were set at 1

[8] We generated a random environment (A,r,o,n) by uniformly choosing a integer value for each variable in the stated range of values.

time unit, as were the duration of information gathering and communication actions; we will demonstrate and discuss the effect of this assumption later in the paper. We used the simulation validation statistic suggested by Kleijnen [Kleijnen, 1987] (where $\hat{y}$ = the predicted output by the analytical model and y = the output of the simulation):

$$z = \frac{y - \hat{y}}{(\text{Var}(y) + \text{Var}(\hat{y}))^{1/2}} \tag{4.10}$$

where $\text{Var}(\hat{y})$ is the predicted variance.[9] The result z can then be tested for significance against the standard normal tables. In each of the 43 cases, we were unable to reject the null hypothesis that the actual mean termination equals the predicted mean termination at the $\alpha = 0.05$ level. For non-statisticians: this is a good thing. The null hypothesis is that our prediction is the same as the actual value, we did not wish to reject it, and we did not. However, such a test has problems since there are other possible reasons which might prevent us from rejecting the null hypothesis. The best approach to avoiding this problem is to report not only the alpha level of the test but also the test's power. With any statistical hypothesis test there are four possible outcomes: we are unable to reject the null hypothesis when it is in reality true , we are unable to reject the null hypothesis when it is in reality false (called a *Type II error*), we reject the null hypothesis when it is in reality false, and we reject the null hypothesis when it is in reality true (called a *Type I error*). The α level is essentially the probability of a Type I error. The *power* of a test is $(1 - \beta)$, where β is essentially the probability of a Type II error (accepting a false null hypothesis). Unfortunately, estimating the power of a test is often difficult, and requires detailed knowledge of the distribution of the test statistic under the condition that the alternative hypothesis is true (this is turn usually requires the computation of the power for several likely alternative hypotheses). Future work will involve the computation of the power for this statistic, or the development of a new test based on other possible statistics that have better-understood power characteristics.

Figure 4.8 shows the mean of 10 repetitions in each environment versus the expected value and its likelihood intervals. The 50% likelihood interval is included to give you some idea of the shape of the underlying distribution; a system designer would of course use the 90% or 95% intervals.

The point of this section is that the analytical model describes the implementation fairly well, and we could use the analytical model to design a good static organization for a given environment, using standard heuristic optimization techniques such as simulated annealing.

4.7.2 Control Costs

The control algorithm presented above is simple and not necessarily optimal. By communicating only when there is no local work to be done, the heaviest-loaded agent gives up the chance for other agents to do the high-level composition in a task group by incrementally transmitting each result (or set of results on a single task group) as it is completed—a maximum potential savings of $(\hat{N} - 1)(a - 1)\mathbf{d}_0(\text{VTM}) + (\hat{N} - 1)\mathbf{d}_0(\text{VCM})$. However, this needs to be balanced with the cost of multiple communication actions, which is $(\hat{N} - 1)\mathbf{d}_0(C)$. Thus

[9] The predicted variance of Equation 4.9 can be easily derived from the statistical identity $\text{Var}(x) = E[x^2] - (E[x])^2$.

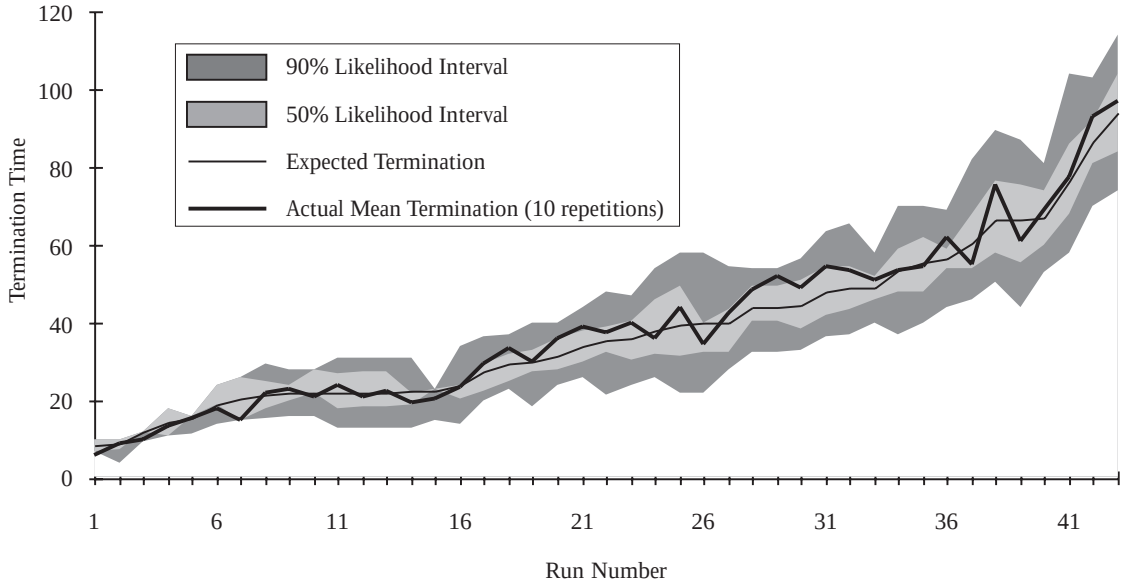

Figure 4.8. Actual system termination versus analytic expected value and analytically determined 50% and 90% likelihood intervals. Runs arbitrarily ordered by expected termination time.

the question of 'when to communicate' (when to incrementally transmit partial results) rests directly on the cost of communication relative to $(a-1)\mathbf{d_0}(\mathrm{VTM})+\mathbf{d_0}(\mathrm{VCM})$ (which depends on both the basic method durations and the agents' organizational structure). Thus, the more agents there are that see a single track the more likely it is that intermediate communication would be useful.

This simple control algorithm can be analyzed easily, unlike many other systems where control costs are ignored. If we view the cost of control as the time spent by an agent when *not* performing an action (executing a method, information gathering, communication), then our algorithm runs in constant time between actions except for the two tests [*get set of currently executable methods*] and [*get set of methods still waiting*]. Each of these in the worst case requires a constant-time test of each element of the full task structure, which is of size $O(\eta r)$. Thus we see how the control costs, too, are related to organizational structure.

4.7.3 Analyzing Dynamic Organizations

In the dynamic organizational case, agents are not limited to the original organization and initial distribution of data. Agents can re-organize by changing the initial static boundaries (changing responsibilities in the overlapping areas), or by shipping raw data to other agents for processing (load balancing). We will assume in this section that the agents do not communicate about the current local state of problem solving directly (see Section 4.8 for thoughts on using meta-level communication). A clearer distinction is that in the dynamic organization each agent makes its initial decision (about changing boundaries or shipping raw data) without access to non-local information. In the meta-level communication situation the agent would have access to both its local information and a summary of the local state of other agents. In

either case the decision to dynamically change the organization is made only once, at the start of an episode.

In the case of reorganized overlapping areas, agents may shift the initial static boundaries by sending a (very short) message to overlapping agents, telling the other agents to do all the work in the overlapping areas. The effect at the local agent is to change its effective range parameter from its static value of $r' = r - o/2$ to some value r'' where $r - o/2 \geq r'' \geq r - o$, changing the first two terms of Equation 4.9, and adding a communication action to indicate the shift and an extra information gathering action to receive the results. Section 4.7.4 discusses a particular implementation of this idea that chooses the partition of the overlapping area that best reduces expected differences between agent's loads and averages competing desired partitions from multiple agents.

In the second case, an agent communicates some proportion p of its initial data to a second agent, who does the associated work and communicates the results back. Instead of altering the effective range and overlap, this method directly reduces the first two terms of Equation 4.9 by the proportion p. The proportion p can be chosen dynamically in a way similar to that of choosing where to partition the overlap between agents (Section 4.7.4).

Whether or not a dynamic reorganization is useful is a function of both the agent's local utility and also the load at the other agent. We will again be concentrating on the agent with the heaviest load. Looking first at the local utility, to do local work under the initial static organization with n task groups, the heaviest loaded agent will take time:

$$S'\mathbf{d}_0(\text{VLM}) + (S' - n)\mathbf{d}_0(\text{VTM}) \tag{4.11}$$

When the static boundary is shifted before any processing is done, the agent will take time

$$\mathbf{d}_0(C_{\text{short}}) + S''\mathbf{d}_0(\text{VLM}) + (S'' - n)\mathbf{d}_0(\text{VTM}) + \mathbf{d}_0(I) \tag{4.12}$$

to do the same work, where C_{short} is a very short communication action which is potentially much cheaper than the result communications mentioned previously, and S'' is calculated using r''. When balancing the load directly, local actions will take time

$$\mathbf{d}_0(C_{\text{long}}) + pS'\mathbf{d}_0(\text{VLM}) + p(S' - n)\mathbf{d}_0(\text{VTM}) + \mathbf{d}_0(I) \tag{4.13}$$

where $\mathbf{d}_0(C_{\text{long}})$ is potentially much more expensive than the communication actions mentioned earlier (since it involves sending a large amount of raw data). If the other agent had no work to do, a simple comparison between these three equations would be a sufficient design rule for deciding between static and either dynamic organization.

Of course, we cannot assume that the other agent is not busy. One solution is to assume the other agent has the *average* amount of work to do. We can derive *a priori* estimates for the average initial work at another agent from Equation 4.9 by replacing the probability function of the max order statistic $g_{a,N,p}(s)$ with the simple binomial probability function $b_{N,p}(s)$. Therefore without any meta-level communication, a system of agents could choose intelligently between static, dynamic overlap reorganization (changing r), and dynamic load balancing (changing p) given these constraints.

4.7.4 Dynamic Coordination Algorithm for Reorganization

This section describes a particular implementation of the general idea described earlier of dynamically reorganizing the partitions between agents for the DSN simulation. This implementation will keep each agent's area of responsibility rectangular, and relaxes competing constraints from other agents quickly and associatively (the order of message arrival does not affect the eventual outcome). To do this, the message sent by an agent requests the movement of the four *corridors* surrounding an agent. The northern corridor of Agent 1, for example, is the northern agent organizational responsibility boundary shared by every agent in the same *row* as Agent 1. As can be seen in Figure 4.9, a 3x3 organization has four corridors (between rows 1 and 2, 2 and 3, and between columns 1 and 2, 2 and 3).

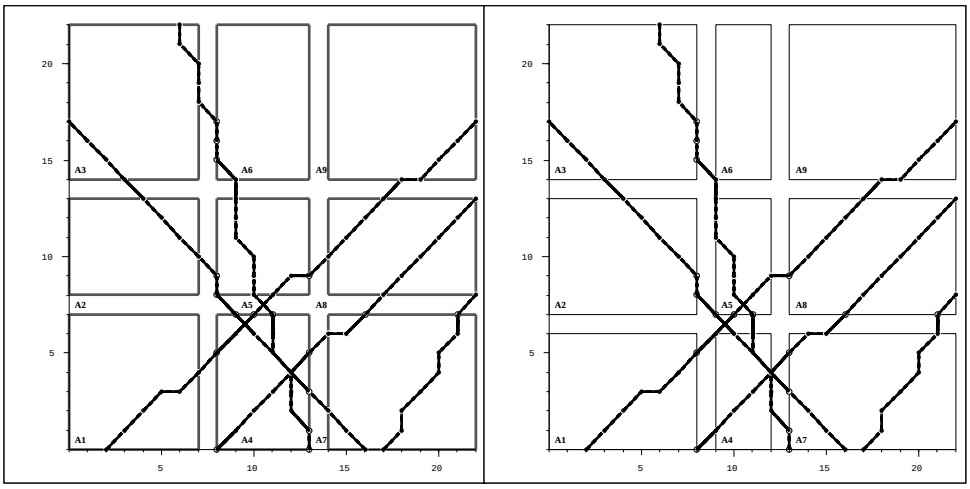

Figure 4.9. On the left is a 3x3 static organization, on the right is the dynamic reorganization result after agents 3, 4, 5 and 7 attempt to reduce their areas of responsibility by one unit. In this example the corridors running North to South have been moved closer by two units to reduce the load on agents 4, 5, and 6 in the second column.

The coordination algorithm described here works with the local scheduling algorithm described earlier in Section 4.7.1. This is consistent with our view of coordination as a *modulating* behavior [Decker and Lesser, 1993a]. The only modification to the local scheduler is that we prevent it from scheduling local method execution actions until our initial communications are completed (the *initial* and *reception* phases, described below).

The coordination algorithm is then as follows. During the *initial* phase the local scheduler schedules the initial information gathering action, and we proceed to the second phase, *reception*. In the second phase we use the local information to decide what organizational design to use, and the parameter values for the design we choose. To do this we calculate the estimated duration of our (known) local work[10], and then estimate that duration under the alternative

[10] The agent knows how much local work it has; in this domain it is computed via Eq. 4.11 given the amount of low-level sensed data that has shown up at the agent.

organizations (dynamic reorganization or load-balancing). When a parameter needs to be estimated, we do so to minimize the absolute expected difference between the amount of work to be done locally and the amount of work done at the remote agent that is impacted the most by the proposed change.

For example, when dynamically restructuring, if the overlap between agents is more than 2 units, we have a choice of reducing the area an agent is responsible for by more than 1 unit (this is the organizational design parameter ρ in question). To decide on the proper reduction (if any), each agent computes its known local work $\hat{W}(\rho)$ using Eq. 4.11 with the actual (not estimated) S' and N computed assuming the agent's area is reduced by ρ. Then the agent finds the value of ρ that minimizes the difference in its known local work $\hat{W}(r-\rho)$ and the *average* work $\bar{W}(r+\rho)$ at the other agent:

$$
\begin{aligned}
S(r,s,N) &= (rs + \frac{r}{2}(N-s)) \\
W(r,s,N) &= S(r,s,N)\mathbf{d}_0(\text{VLM}) + (S(r,s,N) - N)\mathbf{d}_0(\text{VTM}) \\
E[\hat{W}(r)] &= \sum_{N=0}^{n}\sum_{s=0}^{N} g_{A,n,\frac{a}{A}}(N) g_{a,N,p}(s) W(r,s,N) \qquad (4.14) \\
E[\bar{W}(r)] &= \sum_{N=0}^{n}\sum_{s=0}^{N} b_{n,\frac{a}{A}}(N) b_{N,p}(s) W(r,s,N) \qquad (4.15)
\end{aligned}
$$

Equation 4.15 is just a restatement of Eqn. 4.14 (itself derived from Eqns. 4.5, 4.6, 4.7, and 4.11 for the case of the average, not maximally loaded, agent (thus the use of b, the binomial probability function rather than g, the max order statistic p.f.)).

If $\rho = 0$, then the agent will not restructure. If $\rho \neq 0$, then the agent sends a message to all affected agents requesting a reduction of amount ρ in each corridor (north, east, south, and west). The agent sets its current area of interest to include only the unique (non-overlapping) portion of its area (if any), and enters the *unique-processing* phase. During this phase the regular local scheduler described earlier controls method execution actions.

When no more methods unique to this agent can be executed, the coordination algorithm checks the current time. If enough time has passed for the messages from other agents (if any) to arrive (this depends on the communication delays in the system), the coordination algorithm schedules an information-gathering action to retrieve the messages. Note that every agent may reach this point at a different time; agents with a large amount of unique local work may take some time, agents with no work at all will wait idle for the length of communication delay time in the system.

At this point each agent will relax its borders according to the wishes of the other agents. The relaxation algorithm we have chosen is fairly simple and straightforward, though several similar choices are possible. The algorithm is symmetric with respect to the four corridors surrounding the agent, so we will just discuss the relaxation of the northern corridor. There will be a set of messages about that corridor, some wanting it moved up by some amount and some wanting it moved down by some amount—we will consider these as positive and negative votes of some magnitude. The relaxation algorithm sums the votes, and returns the sum unless it is larger than the maximum vote or smaller than the minimum vote, in which case the max or

min is returned, respectively.[11] At this point the agent enters the final *normal processing* phase, and the local scheduler schedules all further actions as described earlier.

4.7.5 Analyzing the Dynamic Restructuring Algorithm

As we did in Section 4.7.1, we can develop an expression for the termination time of any episode where the agents follow this algorithm. To do so, we start with the basic termination time given all of the random variables. This equation is derived from Eqns. 4.14 and 4.15:

$$T(r, \hat{S}, \hat{N}, \bar{s}, \bar{N}) = \max[W(r - \rho, \hat{S}, \hat{N}), W(r + \rho, \bar{s}, \bar{N})] \tag{4.16}$$

where ρ is computed as described in the last section using the given values of $(r, \hat{S}, \hat{N}, \bar{s}, \bar{N})$. To turn this into a predictive formula, we then use the expressions for the probabilities of the terms $\hat{S}$, $\hat{N}$, $\bar{s}$, and $\bar{N}$ (from Eqns. 4.6, 4.7, and 4.5). For example, we can produce an expression for the expected termination of the algorithm:

$$\sum_{\hat{N}=0}^{n} \sum_{\hat{S}=0}^{\hat{N}} \sum_{\bar{N}=0}^{n} \sum_{\bar{s}=0}^{\bar{N}} g_{A,n,\frac{a}{A}}(\hat{N}) g_{a,\hat{N},0.5}(s) b_{n,\frac{a}{A}}(\bar{N}) b_{\bar{N},0.5}(\bar{s}) T(r, \hat{S}, \hat{N}, \bar{s}, \bar{N}) \tag{4.17}$$

We tested the predictions of Equation 4.17 versus the mean termination time of our DSN simulation over 10 repetitions in each of 10 randomly chosen environments. The durations of all tasks were set at 1 time unit, as were the duration of information gathering and communication actions, with the exception of the 4 environments described in the next section. Using the same validation statistic as before (Eq. 4.10) in each case we were unable to reject the null hypothesis that the actual mean termination equals the predicted mean termination at the $\alpha = 0.05$ level.[12]

4.7.5.1 Increasing Task Durations

Figures 4.10 through 4.13 compare the termination of static and dynamic restructuring organizations on identical episodes in four different environments. Ten different episodes were generated for each environment. In order to see the benefits of dynamic restructuring more clearly, we chose task durations for each environment similar to those in the DVMT: $\mathbf{d}_0(\text{VLM}) = 6, \mathbf{d}_0(\text{VTM}) = 2\mathbf{d}_0(\text{VCM}) = 2$.[13] Note that the dynamic organization often does significantly better than the static organization, and rarely does much worse—remember that is many particular episodes that the dynamically organized agents will decide to keep the static organization, although they pay a constant overhead when they keep the static organization (one extra communication action and one extra information gathering action, given that the time for a message to reach all agents is no longer than the communication action time).

---

[11] For example, if 2 agents vote for it to be moved up by one, and two vote for it to be moved up by 2, then the corridor is moved up by only 2, not 6!

[12] The same caveats that we discussed in Section 4.7.1 still apply here.

[13] The idea being that the VLM methods correspond to lowest three DVMT KSIs as a group, and the other methods correspond to single DVMT KSIs, and that a KSI has twice the duration of a communication action.

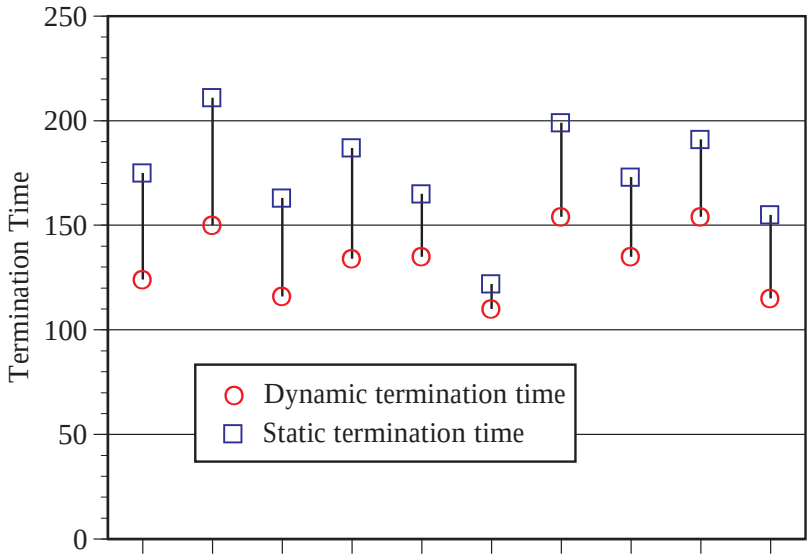

Figure 4.10. Paired-response comparison of the termination of static and dynamic systems the environment $A = 9, r = 9, o = 9, n = 7]$ (ten episodes). Task durations are set to simulate the DVMT (see text).

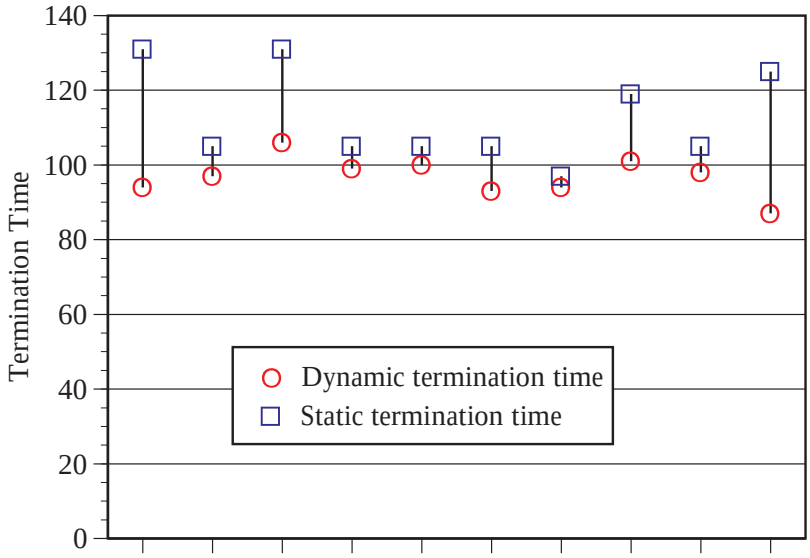

Figure 4.11. Paired-response comparison of the termination of static and dynamic systems the environment $A = 16, r = 8, o = 5, n = 4]$ (ten episodes). Task durations are set to simulate the DVMT (see text).

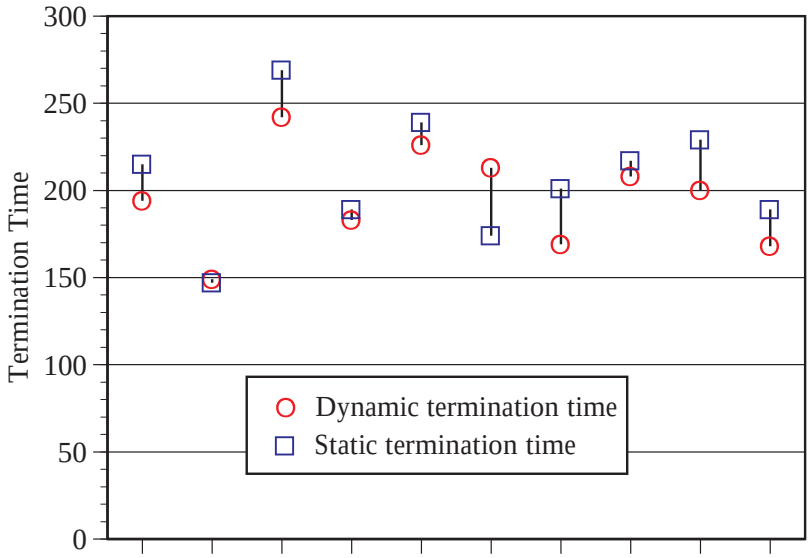

Figure 4.12. Paired-response comparison of the termination of static and dynamic systems the environment $A = 4, r = 9, o = 3, n = 5]$ (ten episodes). Task durations are set to simulate the DVMT (see text).

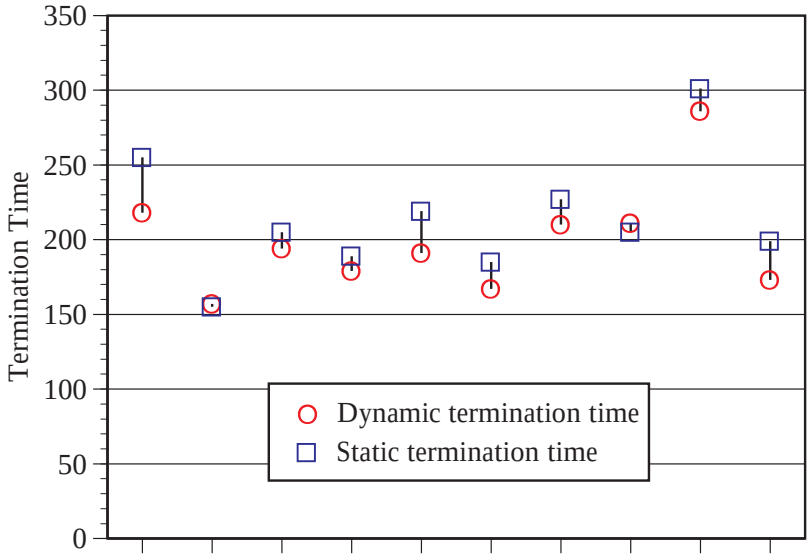

Figure 4.13. Paired-response comparison of the termination of static and dynamic systems the environment $A = 9, r = 10, o = 6, n = 7]$ (ten episodes). Task durations are set to simulate the DVMT (see text).

4.8 Using Meta-Level Communication

For some environments $\mathcal{D} =< A, \eta, r, o, \mathcal{T} >$ one of the three organizational choices (static, reorganized, load balanced) may be clearly better in the long run, but for most environments the choice is not so clear given the variance in system performance. The choice that optimizes performance over the long run is often not optimal in any particular episode. Taking the essential equations for local work in Section 4.7.3, we can compute likelihood intervals on the predicted performance of an organization under each of the three coordination regimes by combining the local likelihood interval on the expected load of the heaviest loaded agent, and the likelihood interval on the average agent load. These results, for the 90% likelihood interval, are shown in Figures 4.14 through 4.19. Again we have assumed that all execution, communication, and information gathering action durations have the same value (making communication relatively expensive). The first figures, Figures 4.14 through 4.16, highlight how the relationship between performance under a static organization and a dynamically load balanced organization changes as the number of agents increases. As expected, load balancing becomes more desirable as the number of agents increases (in relation to the average number of tracks): when there are many agents, the average agent load becomes very low, which offsets the cost of transferring tasks. In this figure the performance difference between static and overlap reorganization remains nearly constant relative to the number of agents.

Figures 4.17 through 4.19 point out how dynamically reorganizing the overlap area increases the performance over static organization as the amount of overlap increases. This result is consistent with Durfee's results [Durfee *et al.*, 1987]. For this graph we assumed that the agents would shrink their entire area of responsibility (as opposed to minimizing the difference in maximum versus average work as described in Section 4.7.4) to try to maximize the visual difference on the graph. This graph shows the need for dynamically calculating the reorganized algorithm shrinkage parameter (ρ) especially at high levels of overlap (note how the dynamically reorganized organization is predicted to do worse at high levels of overlap ($o = 8, 9, 10$) in the $n = 20$ portion). The expected performance difference between the static organization and load balancing remains relatively constant across changing values of o. In both figures we have let $\mathbf{d}_0(C) = \mathbf{d}_0(C_{\text{short}}) = \mathbf{d}_0(C_{\text{long}})$; increasing the differences in these values will move the corresponding curves directly up or down.

The next series of figures demonstrate the effect of the ratio of computation duration to communication duration. This and subsequent figures assume that the dynamic restructuring shrinkage parameter ρ is set to minimize the difference between maximum and average local work as described in Section 4.7.4. Figure 4.20 shows how the expected value and 90% likelihood interval on system termination changes as the duration of a method execution action changes from equal to (1x) a communication action to 10 times (10x) that of a communication action. The task structure remains than of the DSN example described in Section 4.5. In Figure 4.20 we see a clear separation emerge between static and dynamic termination. We did not experiment with changing the communications delay between agents (which is fixed here at 1 time unit) because all data for an episode arrives at once. In continuous environments delay will be more important (see Section 6.1.5 and [Mirchandaney *et al.*, 1989])

These figures assume that the number of task groups n is known beforehand. The reason for this is to highlight the variance implicit in the organization, and minimize the influence of the external environment. Figure 4.21 shows how much extra variance is added when only the

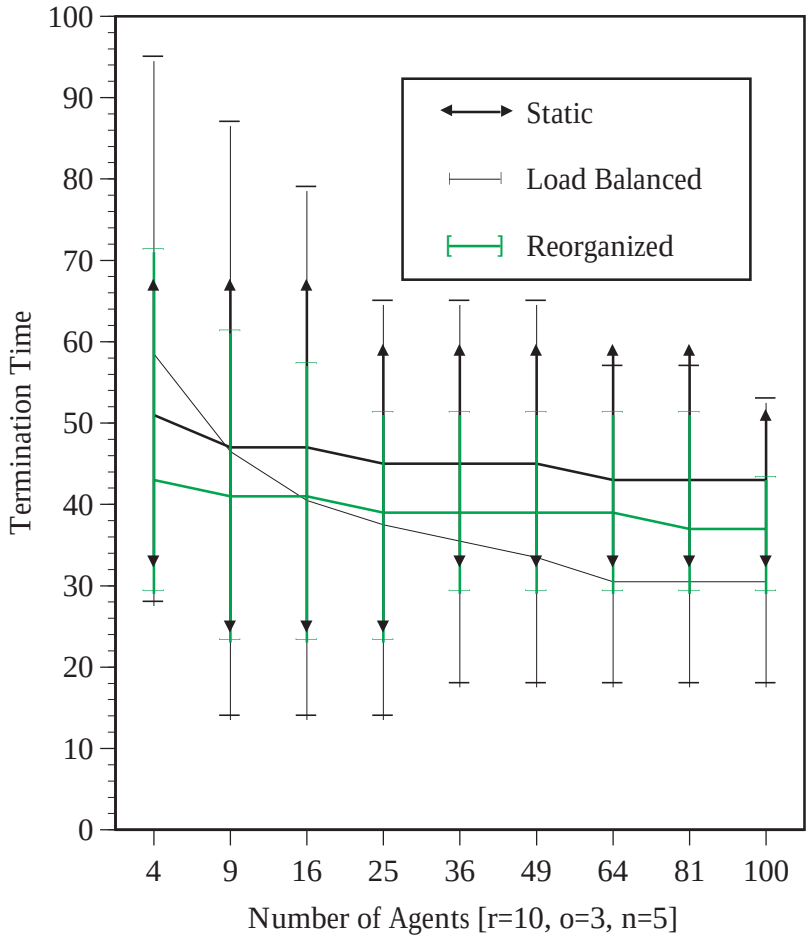

Figure 4.14. 90% likelihood intervals on the expected termination of a system under three coordination regimes, different numbers of agents, and $n = 5$.

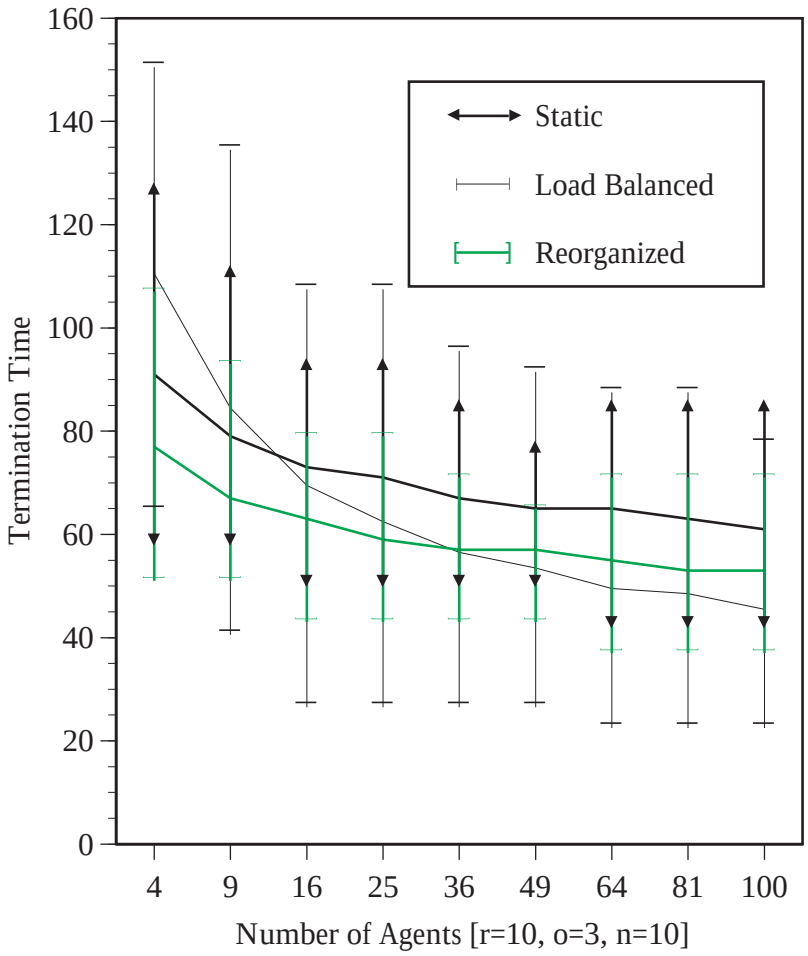

Figure 4.15. 90% likelihood intervals on the expected termination of a system under three coordination regimes, different numbers of agents, and $n = 10$.

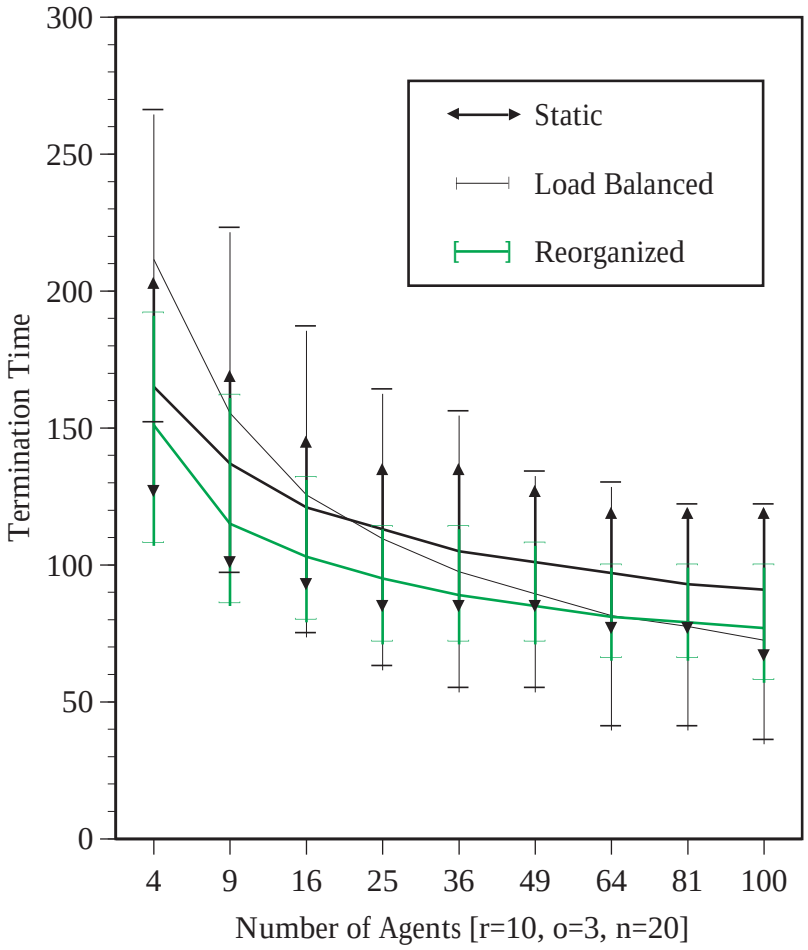

Figure 4.16. 90% likelihood intervals on the expected termination of a system under three coordination regimes, different numbers of agents, and $n = 20$.

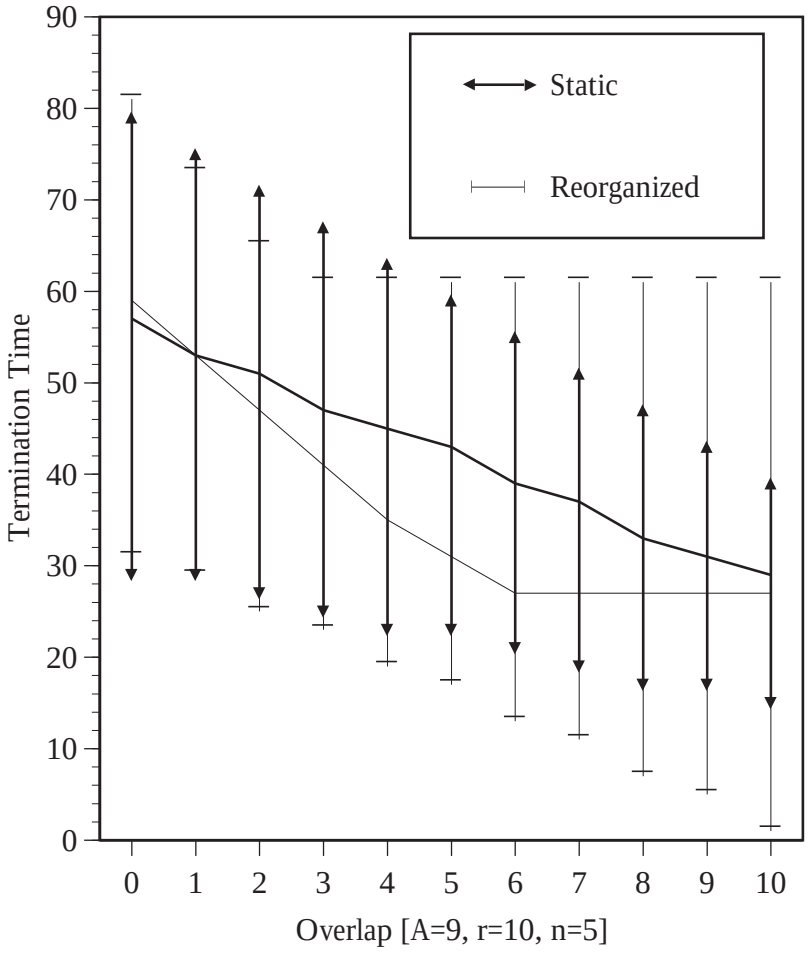

Figure 4.17. 90% likelihood intervals on the expected termination of a system under two control regimes, varying the amount of overlap, with $n = 5$.

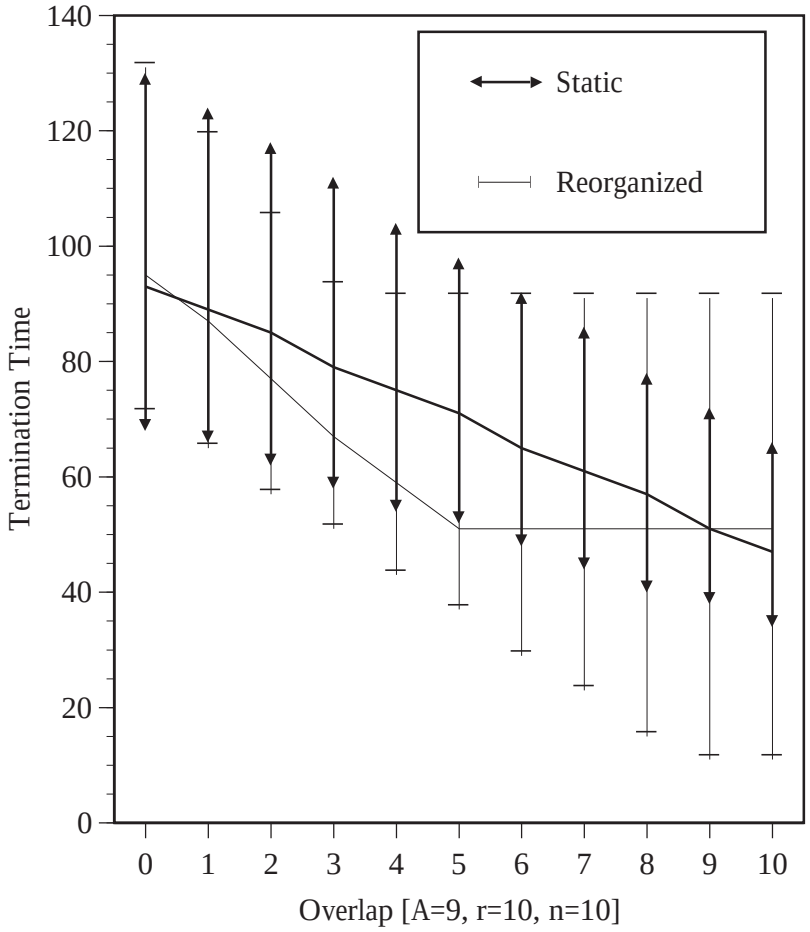

Figure 4.18. 90% likelihood intervals on the expected termination of a system under two control regimes, varying the amount of overlap, with $n = 10$.

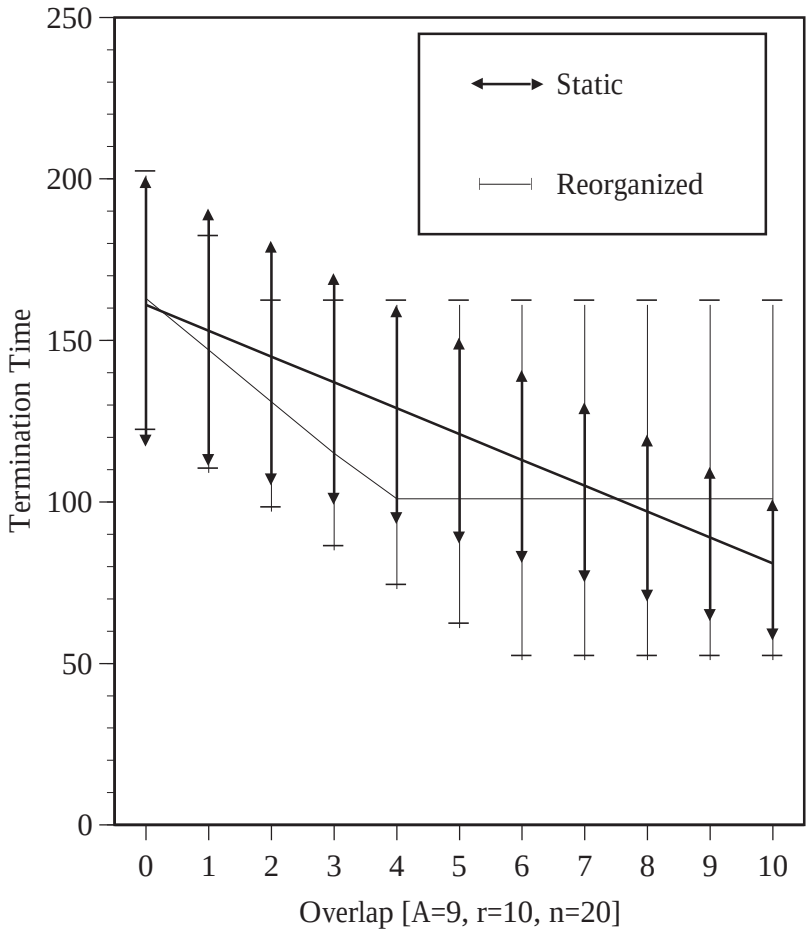

Figure 4.19. 90% likelihood intervals on the expected termination of a system under two control regimes, varying the amount of overlap, with $n = 20$.

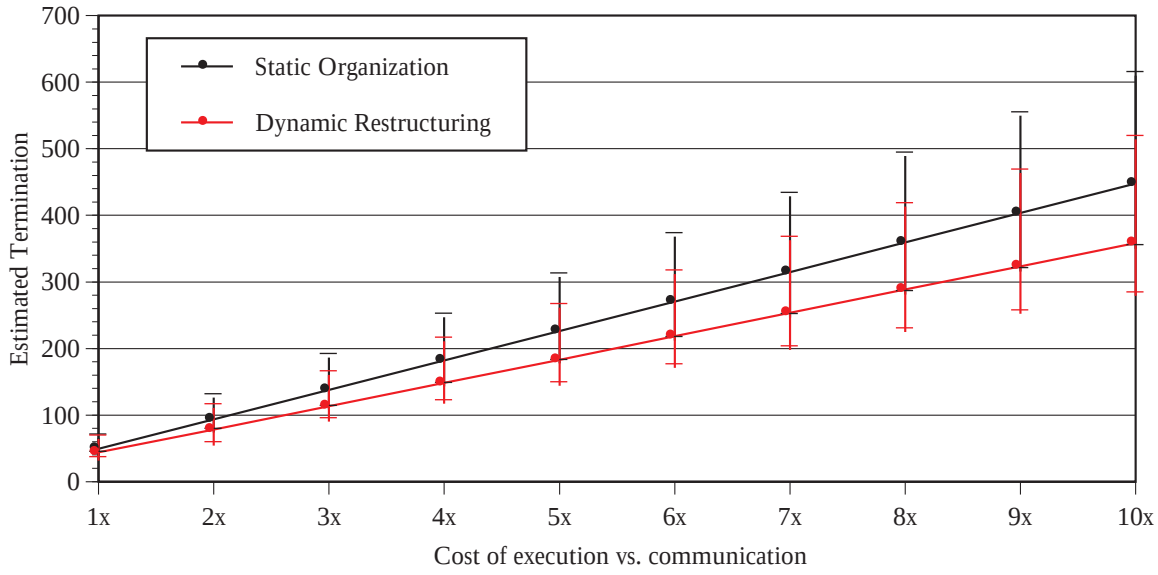

Figure 4.20. Effect of decreasing communication costs on expected termination under a static organization and dynamic restructuring (expected value and 90% likelihood interval, $A = 25, r = 9, o = 9, n = 7$).

expected value of n, which is η, is known. We assume that the number of task groups n (in the DSN example, vehicle tracks) that occur during a particular episode has a Poisson distribution with an expected value of η. The discrete probability function for the Poisson distribution, given in any statistics book, is then:

$$p_\eta(y) \quad = \quad \frac{\eta^y}{y!} e^{-y} \quad [\Pr[n = y]]$$

We can use this probability in conjunction with Eqns. 4.8, 4.11, and 4.15 to calculate the expected value, 50%, and 95% likelihood intervals on termination in the static or dynamic organizations.[14] Note in Figure 4.21 both the large increase in variance when n is random (between the top bar and the third bar in the figure), and more importantly the small decrease in variance in the dynamic restructuring organization (between the first and the second bars). Note also that the mean termination time for the dynamic organization is slightly less than that for the static organization.

These figures bring us to the final point of this chapter: often system performance can be improved significantly by dynamic reorganization, but it will rarely *always* be improved. On *average* the dynamic algorithm does better, especially when computation is more expensive than communication, and assuming that there is some overlapping sensor area with which to trade off computation. Meta-level communication between agents about their local loads should, with a small communication cost, pinpoint the true costs and benefits of the various organizational structures, allowing an informed organizational decision to be made. Instead of an agent making a decision about restructuring or load balancing by assuming the *average* load,

[14] Again, we are providing the 50% interval just to allow the reader to imagine the shape of the underlying distribution better.

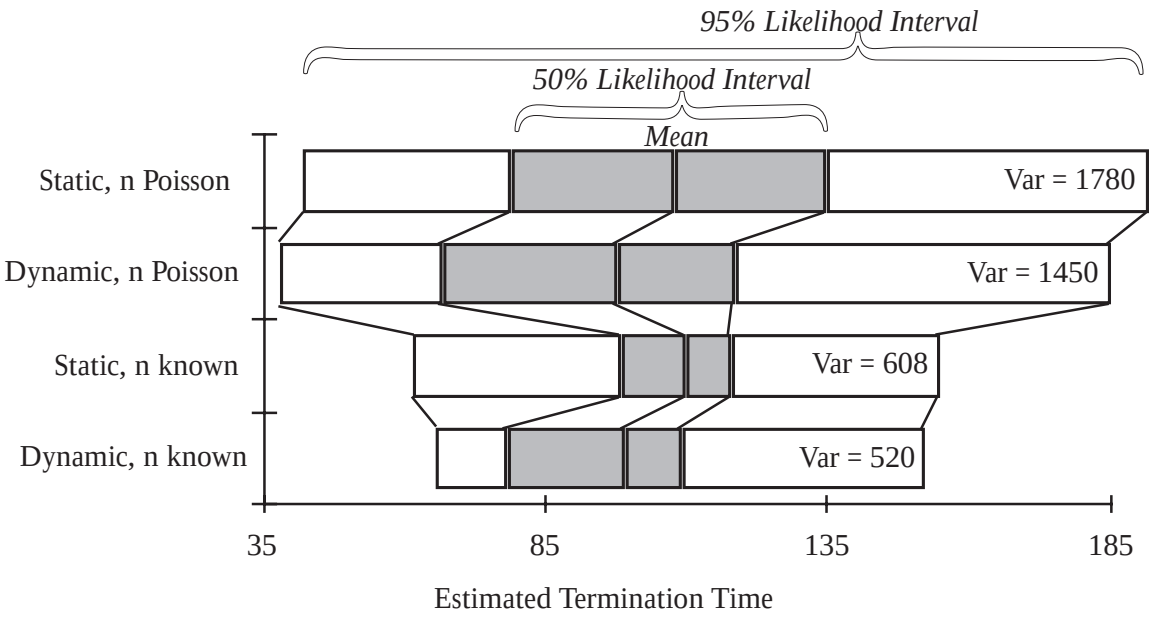

Figure 4.21. Demonstration of both the large increase in performance variance when the number of task groups n is a random variable, and the small decrease in variance with dynamic restructuring coordination $[A = 9, r = 9, o = 9]$. Where n is known, $n = 7$. Where n is a random variable, the expected value $\eta = 7$.

the agent will have the *actual* load for the neighboring agents. As we said in the introduction, the proper organization is often one that exploits *information that resolves uncertainties* about the current environment as it becomes available to the agents, allowing the agents to then create the most efficient organization for the situation.

4.9 Summary

What have I shown in this chapter? I have shown that TÆMS can be used to build models of task environments that can be analyzed mathematically, and in conjunction with simple algorithms for coordinated behavior. In this chapter I showed how variations in termination time could be explained and predicted by environmental variation and the agent's coordination algorithm, a much more satisfying answer than the weaker 'no one algorithm was always the best' given in earlier work [Durfee *et al.*, 1987].

This chapter also demonstrated a methodology for the analysis and design of coordination and scheduling mechanisms, based on MAD. The first step was to build a model of the environment of interest, using TÆMS, in Sections 4.1 through 4.6. At this point the analysis of the endogenous features (such as $\hat{S}$ and $\hat{N}$) of individual episodes, given generative parameters, also took place.

The astute reader will notice that in fact detailed TÆMS structures come in to play for only part of the analysis done here. It is important to point out that even when TÆMS does not give specific tools (like a task structure) for modeling, it still provides a structured method by which modeling can proceed. Thus even though TÆMS does not yet provide us with canned

representations for generative models, it does provide us with, for example, the output of the generative model (some task structure or set of task structures). In this way TÆMS helped us greatly to focus on exactly what should go into the generative model and what was extraneous[15].

TÆMS assumes only a simple view of agents as loci of belief (state) and action. The mathematical analysis in this chapter proceeded with a formal specification of what actions an agent will take based on its current beliefs, using the TÆMS-provided meta-structure for the agent's state-transition function (control, information gathering, communication, and method execution). This chapter looked at two combined control-and-coordination algorithms for agent action (the static algorithm in Section 4.7.1, the one-shot dynamic algorithm in Section 4.7.4 and the meta-level control algorithm in Section 4.7.3).

This chapter then showed the use of the methodology to develop expressions for the expected value of, and likelihood intervals on, the time of termination of a set of agents in any arbitrary simple DSN environment that has a static organizational structure and coordination algorithm. For example, the total time until termination for an agent receiving an initial data set of size $\hat{S}$ is the time to do local work, combine results from other agents, and build the completed results, plus two communication and information gathering actions:

$$\hat{S}\mathbf{d}_0(\text{VLM}) + (\hat{S} - \hat{N})\mathbf{d}_0(\text{VTM}) + (a-1)\hat{N}\mathbf{d}_0(\text{VTM}) + \hat{N}\mathbf{d}_0(\text{VCM}) + 2\mathbf{d}_0(I) + 2\mathbf{d}_0(C) \qquad (4.18)$$

Eq. 4.9 can be used as a predictor by combining it with the probabilities for the values of $\hat{S}$ and $\hat{N}$. I verified this model using the simulation component of TÆMS.

Using this model as a starting point, I extended it to analyze a dynamic, one-shot reorganization algorithm (and have shown when the extra overhead is worthwhile versus the static algorithm). In each case I can predict the effects of adding more agents (Figures 4.14–4.16), changing the relative cost of communication and computation (Figure 4.20), and changing how the agents are organized (in this case, by changing the range and overlap of their capabilities, Figures 4.17–fig:von20). These results were achieved by direct mathematical analysis of the model and verified through simulation in TÆMS. This represents a significant step forward in work on modeling DAI systems.

I specialize MAD by generative model of environmental variance (such as the number of tracks or the load at the most heavily loaded agent). I also demonstrated the importance of environmental variance with respect to agent load balancing and meta-level communication in Section 4.7.5 (I'll return to this question in Section 6.6).

The TÆMS framework for describing complex task environments is intentionally designed to bring together experimental work on complex environments and formal work on the principled construction of agents that act rationally and predictably based on their beliefs, desires, intentions, and goals. I have shown this by applying formal methods to a DSN task with coordination opportunities that are much more complex than single-instance game matrices. The simulation component, which requires the implementation of the ideas, not just abstract analysis, rounds out the picture. At the very least, I hope this approach will allow ideas to be more carefully expressed and thus more approachable by the AI community. I will continue this demonstration in the next chapter through the development and the use of commitments.

In this chapter, stylized information gathering actions are used to represent actions such as 'running external sensors'. The inclusion of explicit information gathering actions, is unique

[15] See also the discussion in Section 3.6.1.

and useful for building models at the 'appropriate' level relative to the question being asked. They are a powerful abstraction mechanism that can replace complex reasoning processes (e.g., planning) with simpler time/computation tradeoffs *for the purpose of modeling*. Often a large, complex system will have many components, not all of which will be under study, or designed. Information gathering actions are used to represent the gross behaviors of these components and anticipate system interactions (or explore possible interactions through perturbation) before and during their design.

Currently there are no characterizations for non-formal coordination algorithms, or for formal techniques in complex environments. This chapter described such a characterization, as will the next.

In the next two chapters, I will move from the simple coordination mechanisms used in these chapters to a family of coordination algorithms constructed from an expandable set of simpler coordination mechanisms. Because these algorithms are more complex and dynamic, I will also move from static, closed-form analyses to simulation-based analyses (but these will use well-characterized environment specifications and many properly generated test episodes—not a few, hand-picked examples).

CHAPTER 5

GENERALIZED PARTIAL GLOBAL PLANNING

> *That is, the relations among what we call our 'actions', 'knowledge', 'beliefs', 'goals', and 'interests' consist of continuous interactions among various structures, mechanisms, traces, impulses, and tendencies that are not necessarily ('naturally' or otherwise) consistent, coordinated, or synchronized and are therefore always* more or less *inconsistent, out-of-phase, discordant, and conflictual.*
>
> *— Barbara Hernstein Smith, Contingencies of Value: Alternative Perspectives for Critical Theory, 1988*

Many researchers have shown that there is no single best organization or coordination mechanism for all environments. This and the following chapters discuss the design and implementation of 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 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 a part. An individual algorithm in the family is defined by a particular set of active mechanisms and their associated parameters. In Chapter 6 I will also 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 [v. Martial, 1992]. The use of commitments in the GPGP family of algorithms is based on the ideas of many other researchers [Cohen and Levesque, 1990, Shoham, 1991, Castelfranchi, 1993, Jennings, 1993]. 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, as I discussed in the overview (Chapter 1), 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. By concentrating on the creation of local scheduling constraints, we avoid the sequentiality of scheduling in the original PGP algorithm [Durfee and Lesser, 1991] that occurs when there are multiple plans. 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, our approach allows consideration of termination issues that were glossed over in the PGP work (where termination was handled by an external oracle). I discuss how to decide when each mechanism should be used and how the mechanisms interact, drawing on simple models of generic task structures and data from simulation experiments where multiple agent teams work in abstract task environments, in the next chapter (Chapter 6).

The way that we specify the family of algorithms in a general, domain-independent way (as responses to certain environmental situations and interactions with a local scheduler) 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.

As an example of how different coordination mechanisms can dramatically effect performance in identical environments, consider Crowston's observation of restaurant service at the TGI Fridays chain [Crowston, 1994]. The task structure and process at any two TGI Fridays restaurants are essentially identical (it is a sit-down restaurant). Crowston discusses two different Fridays locations of equal size. However, the wait time to be seated during periods of heavy use (i.e. the lunch and dinner rush) at the newer of the two locations is reduced to an average of 15 minutes from an hour. The difference between the restaurants is not the structure of their tasks, but the coordination mechanisms. In the old location, enablement relationships between tasks—the table must be bussed before customers can be seated, the food must be cooked and assembled before it can be served—are coordinated by *polling* on the consumer side of the relationship (the host scans for free tables, the waitstaff checks at the kitchen for a table's food order to be up). In the newer, faster restaurant the consumer of an enables relationship is *notified* (a computer screen at the host's station shows free tables and is updated remotely by the cleaning staff as soon as a table is bussed; waitstaff is paged by the kitchen when an order is complete, etc.). Changing the coordination mechanisms has dramatically improved performance during peak times for this organization.

The first section in this chapter will describe the details of the original PGP algorithm that were not covered in Chapter 2, and discuss how we will extend it. The next section will deal in a general way with decisions faced by agents that are coordinating their behavior over a coordination relationship such as facilitates. Section 5.2 will introduce the Generalized Partial Global Planning (GPGP) approach. Sections 5.4 through 5.6 will describe our assumptions

about the agents' architecture. Briefly, we assume each agent has a local scheduler which schedules local actions in an attempt to maximize some utility measure defined on a TÆMS task structure. We also assume that the local scheduler at each agent understands two special types of constraints, called local and non-local commitments, that we will describe in this chapter. We also assume that each agent has a GPGP coordination module and a set of coordination mechanisms. We will describe how the coordination module and the local scheduler interact. Finally, Section 5.7 describes the interface between the local scheduler and the coordination module in more detail.

5.1 Partial Global Planning

Partial global planning [Durfee and Lesser, 1991] was developed as a distributed control technique to insure coherent network problem solving behavior. It is a flexible approach to coordination that does not assume any particular distribution of subproblems, expertise, or other resources, but instead lets nodes coordinate in response to the current situation. Each node can represent and reason about the actions and interactions of groups of nodes and how they affect local activities. These representations are called partial global plans (PGPs) because they specify how different *parts* of the network *plan* to achieve more *global* goals. Each node can maintain its own set of PGPs that it may use independently and asynchronously to coordinate its activities.

A PGP contains an objective, a plan-activity-map, a solution-construction-graph and a status[Durfee and Lesser, 1991]:

- The objective contains information about *why* the PGP exists, including its eventual goal (the larger solution being formed) and its importance (a priority rating or reasons for pursuing it).
- The plan-activity-map represents *what* the nodes are doing, including the major plan steps the nodes are concurrently taking, their costs, and expected results.
- The solution-construction-graph contains information about *how* the nodes should interact, including specifications about what partial results to exchange and when to exchange them.
- The status contains bookkeeping information for the PGP, including pointers to relevant information received from other nodes and when that information was received.

A PGP is a general structure for representing coordinated activity in terms of goals, actions, interactions and relationships.

When in operation, a node's PGPlanner scans its current network model (a node's representation of the goals, actions and plans of other nodes in the system) to identify when several nodes are working on goals that are pieces of some larger network goal (partial global goal). By combining information from its own plans and those of other nodes, a PGPlanner builds PGPs to achieve the partial global goals. A PGPlanner forms a plan-activity-map from the separate plans by interleaving the plans' major steps using the predictions about when those steps will take place. Thus, the plan-activity-map represents concurrent node activities. To improve coordination, a PGPlanner reorders the activities in the plan-activity-map using expectations or predictions about their costs, results, and utilities. Rather than examining all

possible orderings, a PGPlanner uses a hill-climbing procedure to cheaply find a better (though not always optimal) ordering. From the reordered plan-activity-map, a PGPlanner modifies the local plans to pursue their major plan steps in a more coordinated fashion. A PGPlanner also builds a solution-construction-graph that represents the interactions between nodes. By examining the plan-activity-map, a PGPlanner identifies when and where partial results should be exchanged in order for the nodes to integrate them into a complete solution, and this information is represented in the solution-construction-graph.

Why does partial global planning work well in the DVMT? It is because:

- It avoids redundant work among nodes by noticing interactions among the different local plans. Specifically, it notices when two node plans have identical intermediate goals, i.e., when they are generating interpretations of the same region in time and space. This occurs in the DVMT because, in the interest of reliability, nodes have overlapping sensors.
- It schedules the generation of partial results so that they are transmitted to other nodes and assist them at the correct time. To do this it uses the estimates of the times that activities will take and the inferred relation that if node $\mathcal{A}$ estimates that it will take less time than node $\mathcal{B}$ to complete an intermediate goal, and the goals are spatially near, that node $\mathcal{A}$ can provide facilitative information to node $\mathcal{B}$.
- It allocates excess tasks from overloaded nodes to idle nodes. Node plans provide the information needed to determine if a node is overburdened or underutilized. A node is underutilized if it is either idle or participates in only low-rated PGPs. A node is overburdened if its estimated completion time of a subgoal of goal G is much later than the completion time of all the other subgoals of G [Durfee and Lesser, 1989].
- It assumes that a goal is more likely to be correct if it is compatible with goals at other nodes. In the DVMT task, a goal represented a processing task to ascertain whether a vehicle was moving in a region r at time t. This goal could, in fact, be wrong—based on noise or errorful sensor data that was the basis for the preliminary task analysis that generated the goal. Nodes choose local plans to work on based on the highly rated PGPs they have received. Thus, if the intermediate goals of a node become part of a PGP, then they are worked on before other intermediate goals in other local plans the node may have (even though the node may have rated those local plans higher in its local view).
- It is terminated externally when it first reaches the correct solution, rather than through some internal termination criteria.

The partial global planning framework lets nodes converge on common PGPs in a stable environment (where plans do not change because of new data, failed actions, or unexpected effects of their actions). When network, data, and problem-solving characteristics change and communication channels have delay and limited capacity, nodes can locally respond to new situations, still cooperating but with potentially less effectiveness because they have somewhat inconsistent PGPs [Durfee and Lesser, 1988a]. The PGP framework does not deal with conflicts in physical resources.

The original PGP algorithm orders intermediate 'goals' according to their cost as computed from the cross-product of a vector of seven computable factors (described below) and global cooperation parameters that give a weight to the corresponding term in the calculation. In

our approach, this information has been captured symbolically by the task structure. The PGP 'goals' were communicated at a fixed level of detail, and corresponded directly to tasks or activities. We could re-implement most of the original PGP algorithm by using a hill-climbing local scheduler, and assigning a priority according to the original PGP evaluation function—however, we are instead using a more sophisticated local scheduler that can understand more complex constraints such as delays and deadlines. The Durfee PGP hill climbing scheduler worked by computing a 'cost' for each activity/goal in the plan, and trying to order the plan so that the least expensive activities were earlier. There were seven factors that went into computing the cost for each plan activity. The relationship of each of the seven factors used in the original PGP evaluation function to our new representations is as follows [Note that these terms are the terms used in Durfee's thesis, and differ considerably from the usage of these terms in work on distributed operating systems]:

PGP Redundancy: Was defined by Durfee as the number of nodes that can perform this goal/activity/task. Thus, a plan activity with a high 'redundancy' score would be moved later in the plan. A GPGP local scheduler can use the equality and subtask relationships in the TÆMS task structure to determine redundancy.

PGP Reliability: Was defined by Durfee as the number of nodes that *cannot* perform this goal. This is the inverse of Durfee's definition of PGP redundancy.

PGP Duration: Was defined by Durfee as the duration of the goal/activity/task. Obviously a GPGP local scheduler can obtain this measure from a TÆMS task structure as well.

PGP Predictiveness: Durfee defined this as follows. Any goal with a duration of x could provide predictive information for a goal of duration y if $x < y$. The predictiveness measure was then the minimum activity distance (minimum number of sensed times) between the two goals. Obviously this definition is only useful for the DVMT. GPGP generalizes this concept to the *facilitates* relationship or similar relationships and considers it to be a domain-dependent relationship.

PGP Locally-predicted: Was defined by Durfee as the minimum activity distance (minimum number of sensed times) between a goal and any goal to be executed before it. The effect of this measure was to keep a node from jumping around between times in constructing a track; extending an existing partial solution was preferred. A GPGP local scheduler has available the subtask-oriented, hierarchical construction of the task structure. GPGP doesn't require absolute stability in the local schedule as long as the commitments made to other agents are met.

Independence: Was defined by Durfee as how many goals occurred before a given goal in the initial local node plan. This measure was intended to keep the PGPlanner from straying too far from the original local node plan ordering—goals that were originally late in the ordering would tend to stay late in the PGP ordering. The independence measure for each goal is constant, because it depends only on the initial local node plan, not on the position of the goal in the re-ordered plan. Goals later in the initial ordering have higher independence measures. Because the PGP algorithm used a swapping procedure to create a new ordering from the old, the higher independence measure made later goals harder to swap. A GPGP local scheduler receives local ordering preferences, and so it knows what local orderings were necessary, which may be negotiated, and which are only

preferences. Because the local scheduler is not supplanted by a separate PGPlanning scheduler, this relationship is not directly needed.

Diversity: Durfee defined the diversity value of a goal as 0 if it did not derive redundant information, and also 0 if all goals following it derived redundant information. Otherwise, the diversity of a goal was measured by the minimum activity distance between the goal and the later non-redundant goals. The effect was to plan to do non-redundant work before redundant work. The GPGP coordination mechanisms detect redundancy through coordination relationships instead, so this DVMT-specific measure is not needed.

5.1.1 Issues in Extending the PGP Mechanisms

5.1.1.1 Heterogeneous Agents

How can the PGP mechanisms be extended to handle agents that have different local problem solving criteria? This can arise in several ways:

- Some agents in the system are humans with local (personal) decision criteria that cannot be adequately or fully modeled.
- Some agents in the system have different expertise, and hence different local decision criteria (cooperative design problems [Lander and Lesser, 1989], pilot's associate-style problems [Smith and Broadwell, 1987]). The PA scenario in Section 3.6.3.4 is a classic example of heterogeneous agents with shared global goals and differing local expertise.

The PGP mechanism assumes a shared local and global decision evaluation function (so that all agents, given the same information and enough time, will arrive at the same decisions). Conflict between agents comes about because some agents lack data or have out-of-date data. Agents do not have to exchange or negotiate about decision criteria. While this well-documented assumption simplified the PGP mechanism, a homogeneous agent assumption (where the local decision criteria are shared) is not always appropriate. The PGP mechanism also assumes that the agents will pursue one goal at a time — the goals are ordered, and if an agent has excess capacity it can fill it with tasks from lower-rated goals. Planning for the simultaneous achievement of multiple goals is not supported.

The modularity of the PGP mechanism (which separates the local agent's incremental planner [Durfee and Lesser, 1988b] from the PGPlanner) comes close to permitting heterogeneous local decision criteria. The only problem arises when the PGPlanner reorders the node plan for another agent. The PGP plan evaluation function that was used to develop a global schedule contains terms to avoid upsetting the order of another agent's plan (*independence* measured the distance of the current ordering from the original node plan ordering, *locally-predicted* measured the distance of the current ordering from regular time order). In some domains a portion of this ordering may be fixed. We will suggest marking scheduling commitments as *hard*, *negotiable*, and *soft*. This allows the local scheduler to rule out certain impossible orderings (hard constraints), and to avoid those that may cause replanning at the target node (negotiable constraints). More importantly, we advocate a separation of coordination mechanisms and local scheduling mechanisms, so that nodes do not work out complete schedules for other nodes, but rather pass only scheduling constraints.

5.1.1.2 Dynamic Agents

How can the mechanisms be extended to handle agents that have a great deal of latitude in the methods that they use to solve problems? Each method may have a different effect on the characteristics of the solution, such as completion time or certainty. These agents can appear in human systems and systems where agents use approximate processing or anytime algorithm techniques [Decker *et al.*, 1990, Boddy and Dean, 1989]. In the PA scenario in Section 3.6.3.4, the mission planner might solve its dilemma by using different algorithms to respond to each plan request. A fast but inaccurate algorithm may suffice to give the pilot an idea of a corridor of escape, while a more complex and precise algorithm can be given the bulk of the computational resources with which to plan the near-term tactical maneuver.

Because only one method existed for accomplishing a goal (and no set of different criteria existed for determining what would be considered an acceptable solution), the PGP mechanism could equivalently exchange goals and the plans to accomplish those goals, at a single level of detail. The node plans that were exchanged indicated the goal of an agent to produce a track with certain characteristics (classes, sensed times, and regions) and a plan consisting of the ordering of the sensed times at which the agent would work (called *i-goals*), expected i-goal durations, and a mapping of the i-goal start and end times with respect to node problem solving time.

Two extensions need to be made. First, communicating tasks at a single level of detail is inappropriate and potentially wasteful in more complex domains[Knoblock, 1991]; certainly the detection of the interactions of two tasks will not always be simple [Robinson and Fickas, 1990, v. Martial, 1992, Durfee and Montgomery, 1991]. Secondly, many different methods may exist for a task, each with its own effects on duration, precision, and other task characteristics. This makes the existing PGP node plan structure change rapidly when problem solving methods are changing dynamically (as an agent reacts to the problem being solved). The node plan structure can be modified to hold ranges as well as a best current estimate for a value, but it is also likely that agents will have to reason and perhaps negotiate about predictability versus reliability issues as well [Durfee and Lesser, 1988a]. The node plan structure could also be expanded with contingency plans for "routine expectation failures" [Dean, 1987] to allow for predictability in the face of a changing environment. Agents can also make commitments to certain task characteristics, and add explicit slack to schedules[Decker and Lesser, 1993a].

5.1.1.3 Real-time Agents

What happens when time becomes an integral part of local and shared goals? Dynamic agents will be able to modify both task durations (perhaps trading them off for other task characteristics) *and* the task deadlines themselves[Decker *et al.*, 1992, Garvey and Lesser, 1993]. In the PA scenario in Section 3.6.3.4, the mission planner's dilemma arises from the fact that it is under real-time constraints — if there were no impending deadlines for the pilot and tactical planner, the mission planner would have little reason to prefer one allocation of its computational resources over another.

While the PGP mechanism estimated the times for tasks or goals to be completed in order to spot idle processing resources, it did not handle deadlines. I-goals had expected durations; node-plans anchored (mapped) the completion of the various i-goals to a plan activity map.

Experiments were conducted with the local incremental planner that did indicate the ability to plan to meet deadlines in a single agent [Lesser *et al.*, 1988, Decker *et al.*, 1990].

In extending the architecture to so-called "real-time" problem-solving, agents may have goals with hard deadlines, which add constraints to the construction of a plan activity map. Furthermore, the addition of hard deadlines or other domain constraints changes the nature of the interaction between a node's local problem solving mechanism and the PGP mechanism — some of the local ordering will remain local preference but some may be due to hard constraints, as discussed in Section 5.1.1.1 above. The existing hill-climbing algorithm for scheduling may no longer be appropriate, and is one reason why we now advocate the separation of coordination and local scheduling. Our initial experiments with a real-time local scheduler[Decker and Lesser, 1993a] show that a large part of distributed real-time performance emerges from sophisticated local real-time scheduling capabilities[Garvey and Lesser, 1993].

Often in real time situations planning is *reactive*, where the current situation mostly controls an agent's actions (where the "current situation" may include both local and global information), rather than *reflective*, where a sequence of actions is planned out in some detail before execution. This is because the agent must respond quickly, but more importantly, the agent may be too uncertain of the outcomes of its actions and of the changing world state to plan too far into the future. However, an intelligent agent will make use of periodic tasks, which occur in a predictable fashion, and known non-periodic tasks, to build a opportunistic planning framework that can keep an agent from painting itself into a corner with purely reactive planning techniques, or from exhaustively planning uncertain future details with reflective planning techniques[Decker *et al.*, 1992, Garvey and Lesser, 1993].

5.2 Generalizing the Partial Global Planning Mechanisms

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) [Lawrence and Lorsch, 1967, Galbraith, 1977, Stinchcombe, 1990] and cooperative distributed problem solving [Fox, 1981, Durfee *et al.*, 1987, Durfee and Montgomery, 1991, Decker and Lesser, 1993b]. 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.

So far we have made two important statements: that there is ample evidence that no single coordination algorithm is 'the best' across different environments, and that we are going to describe a *family* of coordination algorithms. That we need a family of algorithms follows from the first statement; we will now briefly discuss how that family is organized. At the most abstract level, each of the five mechanisms we are about to describe are parameterized independently

(the first two have three possible settings and the last three can be in or out) for a total of 72 combinations. Many of these combinations do not show significantly different performance in randomly generated episodes, as will be discussed in the experiments (Section 6), although they may allow for fine-tuning in specific applications. More mechanisms can (and have) been added to expand the family, but the family can also be enlarged by making each mechanism more situation-specific. For example, mechanisms can have their parameters set by a mapping from dynamic meta-level measurements such as an agent's load or the amount of real-time pressure. Mechanisms can be 'in' or 'out' for individual *classes* of task groups, or tasks, or even specific coordination relationships, that re-occur in particular environments. The cross product of these dynamic environmental cues provides a large but easily enumerated space of potential coordination responses that are amenable to the adaptation of the coordination mechanisms over time by standard machine learning techniques.[1] In the experimental section of dissertation (Chapter 6) we will only consider the coarsest parameterization of the mechanisms. The reader should keep in mind when reading about the various coordination techniques that we have explicitly stated that they will not be useful in every situation. Instead, our purpose is to carefully describe the mechanisms and their interactions so that a decision about their inclusion in an application can be made analytically or experimentally (we will present an example in Section 6).

5.2.1 Uncertainty

Less uncertainty in the environment means less uncertainty in the existence and extent of the task interdependencies, and less uncertainty in local scheduling—therefore the agents need less complex coordination behaviors (communication, negotiation, partial plans, etc) [Lesser, 1991]. For example, one can have cooperation without communication [Genesereth *et al.*, 1986] if certain facts are known about the task structure by all agents. In this chapter and the next, agents will not have *a priori* information about the structure of an episode, but they will be able to get information about a subset of the structure after the start of problem solving—no single agent working alone will be able to construct a view of the entire problem (task structure) facing the group. This lack of information (another form of environmental uncertainty) can cause the local scheduler to make sub-optimal decisions. Some of this uncertainty will arise from disparities in the objective (real) task structure and the subjective (agent-believed) task structure. For example, an agent may not know the true duration of a method and if the execution variance is high may not even have a good estimate. In this chapter and the next we will be looking at environments that do not have this characteristic (of large variance between objective characteristics and subjective estimates of those characteristics). Instead we will focus on a second source of structural uncertainty—each agent has only a partial subjective view of the current episode.

5.2.2 Task Interrelationships

Task interrelationships include the relationships of tasks to the performance criteria by which we will evaluate a system, to the control decision structures of the agents which make

[1] Each mechanism may be key for some environment—remember, no set of mechanisms will be the best for all environments.

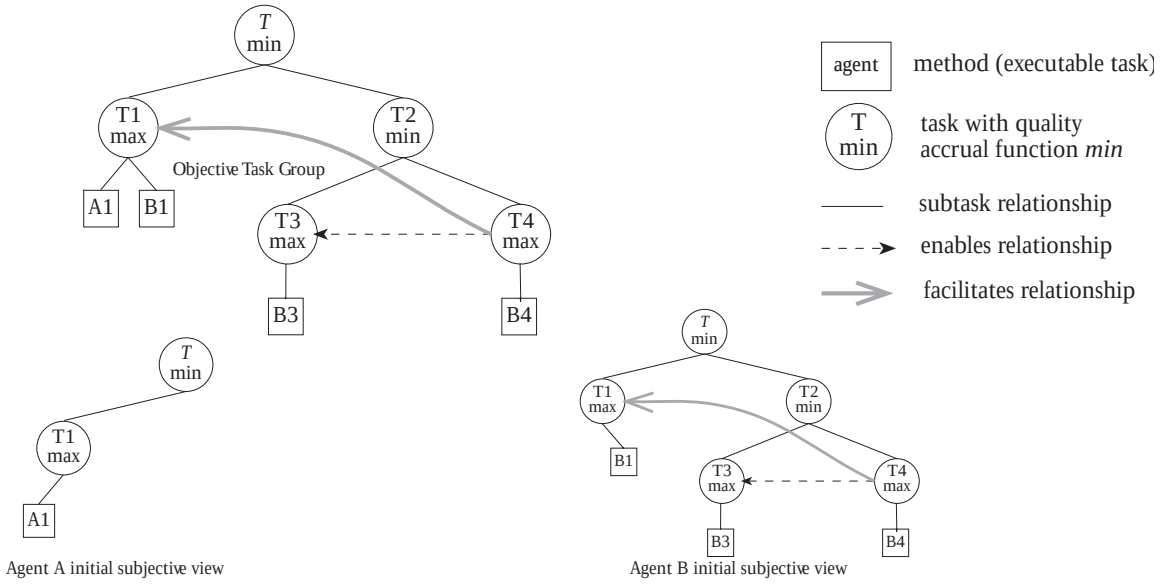

Figure 5.1. Agent A and B's subjective views (bottom) of a typical objective task group (top)

up a system, and to the performance of other tasks. We will represent, analyze, and simulate the effects of task interrelationships using the TÆMS. The TÆMS objective subtask relationship indicates the relationship of tasks to system performance criteria; the subjective mapping indicates the initial beliefs available to an agent's control decision structures; various non-local effects such as enables and facilitates indicate how the execution of one task affects the duration or quality of another task. When a relationship extends between parts of a task structure that are subjectively believed by different agents, we call it a *coordination relationship*. The basic idea behind Generalized Partial Global Planning (GPGP) is to detect and respond appropriately to these coordination relationships. A key point is that coordination relationships are abstractly defined in a domain-independent way, so we can catalogue coordination strategies and link them to potential features of a task environment. Figure 5.1 shows an objective task group and agent A's subjective view of that task group. This notation was formally defined in Chapter 3.

5.3 Generalized Partial Global Planning: Conceptual Overview

This section will provide a quick overview of the GPGP approach. Figure 5.2 shows a simple two-agent example that we will use. Each agent has as part of its architecture a belief database and local scheduler as described back in Chapter 3. The local scheduler uses the information in the belief database to schedule method execution actions for the agent in an attempt to maximize its performance. We add to this a coordination module that is in charge of communication actions, information gathering actions, and in making and breaking *commitments* to complete tasks in the task structure. The coordination module consists of several coordination mechanisms, each of which notices certain features in the task structures in the belief database, and responds by taking certain communication or information gathering actions, or by proposing new commitments. The coordination mechanisms rest in a shared coordination module substrate that keeps track of local commitments and commitments

received from other agents, and that chooses from among multiple schedules if the local scheduler returns multiple schedules.

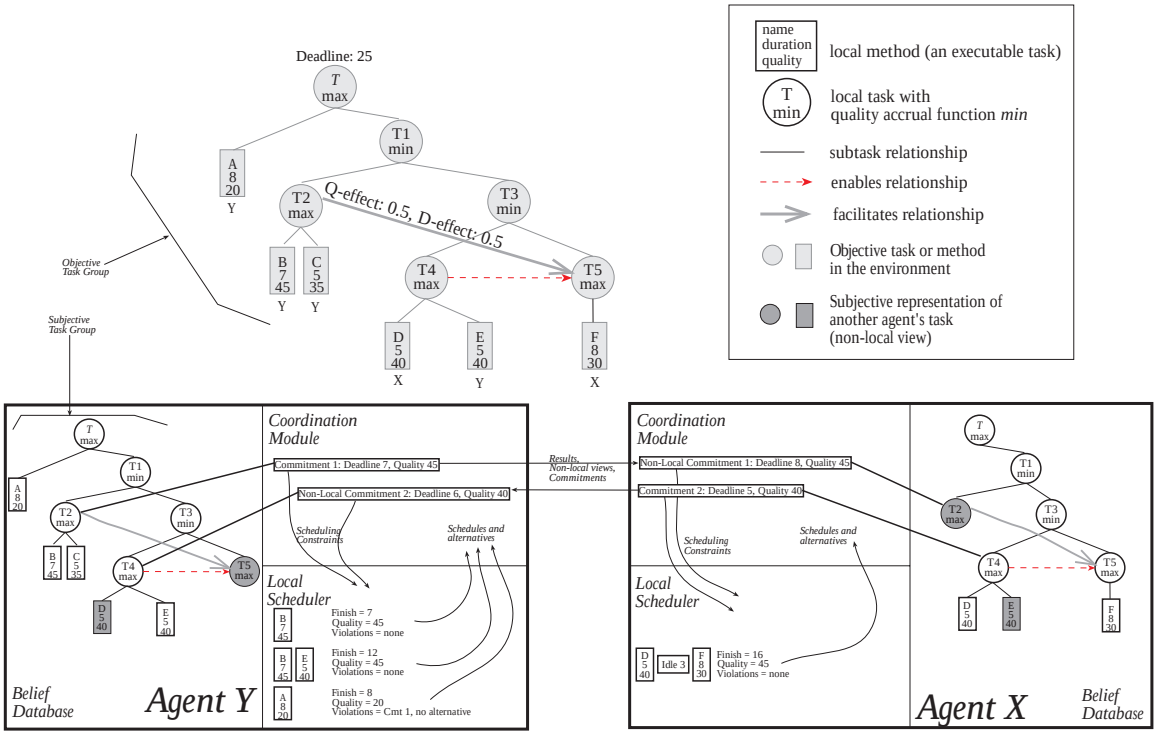

Figure 5.2. An Overview of Generalized Partial Global Planning

Here is a short example intended only to give the reader a feel for the overall approach. The details will be covered in the body of this chapter. In Figure 5.2, both agents have executed an initial information gathering action, and have their initial views of the task structure (everything in the agents' belief database *except* for the shaded tasks (Tasks 2, 5, D and E), and the relationships touching the shaded tasks). One of the coordination mechanisms (Mech. 1, update non-local views) performs an information gathering action to determine which tasks may be related to tasks at other agents ("detect coordination relationships"). These tasks are then exchanged between the agents, resulting in the belief databases shown in the figure (including the shaded tasks). Other mechanisms react to the task structure. One mechanism (Mech. 5, handle soft predecessors) notices that Task 2 at Agent Y facilitates Task 5 at Agent X. In order that Agent X might schedule to take advantage of this, Agent Y's mechanism makes a local intermediate deadline commitment to complete its Task 2 by time 7 with minimum quality 45 (you and I may infer that Y intends to execute Method B, but that local information is not a part of the commitment). A commitment is made in two stages: first it is made locally to see if it is possible as far as the agent's local scheduler is concerned, and then it is made non-locally and communicated to the other agents that are involved. Note that the deadline on the non-local version of this commitment is later (time 8) to take into account the communication delay

(here, 1 time unit). Similarly, Agent X has a mechanism (Mech. 3, handle simple redundancy) that notices that either agents X or Y could do Task 4. Agent X does eventually commit to this task (the process is a bit more complicated as will be explained later) and communicates this commitment to Agent Y.

In both cases the agents' local schedulers use the information about the task structure they have in their belief database, and the local and non-local commitments, to construct schedules. The local scheduler may return multiple schedules for several reasons we explain later. Each schedule is evaluated along the dimensions of the performance criteria (such as total final quality and termination time) and for what (if any) local commitments are violated. If a commitment is violated, the local scheduler may suggest an alternative (for instance, relaxing a quality or intermediate deadline constraint). The coordination module chooses a schedule from this set, and handles the retraction of any violated commitments. The rest of this chapter will describe the agents, local scheduler, coordination module, and coordination mechanisms in more detail.

5.4 The Agent Architecture

As we have mentioned several times in this dissertation, the TÆMS framework makes very few assumptions about the architecture of agents (agents are loci of belief and action). Agents have some control mechanism that decides on actions given the agent's current beliefs. There are three classes of actions: method execution, communication, and information gathering. Method execution actions cause quality to accrue in a task group (as indicated by the task structure). Communication actions are used to send the results of method executions (which in turn may trigger the effects of various task interrelationships) or meta-level information. Information gathering actions add newly arriving task structures, or new communications, to an agent's set of beliefs.

Formally, we write $B_A^t(x)$ to mean agent A subjectively believes x at time t. We will shorten this to $B(x)$ when writing about any agent's beliefs or when the agent and/or time is clear by context (from Shoham[Shoham, 1991]). An agent's subjective beliefs about the current episode $B(\mathbf{E})$ includes the agent's beliefs about various task groups (e.g., $B(\mathcal{T}_i \in \mathbf{E})$), and an agent's beliefs about each task group includes beliefs about the tasks in that task group (e.g., $B(T_a, M_b \in \mathcal{T}_i)$) and the relationships between these tasks (e.g., $B(\mathsf{enables}(T_a, M_b))$).

The GPGP family of coordination algorithms makes stronger assumptions about the agent architecture. We briefly described this architecture in Figure 5.2. Most importantly, it assumes the presence of a local scheduling mechanism (to be described in the next section) that can decide what method execution actions should take place and when. It assumes that agents do not intentionally lie and that they believe what they are told (i.e. if agent $A1$ tells agent $A2$ at time t_1 with communication delay δ that $B_{A1}(\mathsf{enables}(T_a, M_b))$, then $B_{A2}^{t_2}(\mathsf{enables}(T_a, M_b))$ where $t_2 \geq t_1 + \delta$ is the earliest time after the communication arrives that agent $A2$ performs a new communication information gathering action to read the message from $A1$). However, because agents can believe and communicate only subjective information, they may unwittingly transmit information that is inconsistent with an objective view (this can cause, among other things, the phenomena of *distraction*). Finally, the GPGP family approach requires domain-dependent code to detect or predict the presence of coordination relationships in the local task structure [Decker *et al.*, 1991]. In this dissertation we have referred to that domain-dependent code as the information gathering action *detect-coordination-relationships*; we will describe this action more in Section 5.6.2.

5.5 The Local Scheduler

Each GPGP agent contains a local scheduler that takes as input the current, subjectively believed task structure and produces a schedule of what methods to execute and when. Using the information in the subjective structure about the potential duration, potential quality, and relationships of the methods, it chooses and orders executable methods in an attempt to maximize a pre-defined utility measure for each task group $\mathcal{T}$. In this chapter and the next, we use Garvey's Design-To-Time local scheduler, where the utility function is the sum of the task group qualities. The local scheduler attempts to maximize this utility function $\mathrm{U}(\mathbf{E}) = \sum_{\mathcal{T} \in \mathbf{E}} Q(\mathcal{T}, \mathrm{D}(\mathcal{T}))$, where $Q(T,t)$ denotes the quality of T at time t as defined in [Decker and Lesser, 1993d].[2]

Beside the subjective task structure, the local scheduler should accept a set of *commitments* $\mathbf{C}$ from the coordination component. These commitments are extra constraints on the schedules that are produced by the local scheduler. For example, if method 1 is executable by agent A and method 2 is executable by agent B, and the methods are redundant, then agent A's coordination mechanism may *commit* agent A to do method 1 both locally and socially (commitments are directed to particular agents in the sense of the work of Shoham and Castelfranchi [Castelfranchi, 1993, Shoham, 1991]) by communicating this commitment to B (so that agent B's coordination mechanism records agent A's commitment, see the description of non-local commitments $\mathbf{NLC}$ below). The implementation in this dissertation will use two types of commitments: $C(\mathsf{Do}(T,q))$ is a commitment to 'do' (achieve quality for) T and is satisfied at the time t when $Q(T,t) \geq q$; the second type $C(\mathsf{DL}(T,q,t_{dl}))$ is a 'deadline' commitment to do T by time t_{dl} and is satisfied at the time t when $[Q(T,t) \geq q] \wedge [t \leq t_{dl}]$.[3]

A schedule S produced by a local scheduler will consist of a set of methods and start times: $S = \{\langle M_1, t_1\rangle, \langle M_2, t_2\rangle, \ldots, \langle M_n, t_n\rangle\}$. The schedule may include idle time, and the local scheduler may produce more than one schedule upon each invocation in the situation where not all commitments can be met. The different schedules represent different ways of partially satisfying the set of commitments (see [Garvey *et al.*, 1994], the next section, and Section 5.7). The function $\mathrm{Violated}(S)$ returns the set of commitments that are believed to be violated by the schedule. For violated deadline commitments $C(\mathsf{DL}(T,q,t_{dl})) \in \mathrm{Violated}(S)$ the function $\mathrm{Alt}(C,S)$ returns an alternative commitment $C(\mathsf{DL}(T,q,t^{\star}_{dl}))$ where $t^{\star}_{dl} = \min t$ such that $Q(T,t) \geq q$ if such a t exists, or NIL otherwise. For a violated Do commitment an alternative may contain a lower minimum quality, or no alternative may be possible.

The final piece of information that is used by the local scheduler is the set of non-local commitments made by other agents $\mathbf{NLC}$. This information can be used by the local scheduler to coordinate actions between agents. For example the local scheduler could have the property that, if method M_1 is executable by agent A and is the only method that $\mathsf{enables}$ method M_2 at agent B (and agent B knows this $B_B(\mathsf{enables}(M_1, M_2))$), and $B_A(C(\mathsf{DL}(M_1, q, t_1))) \in B_B(\mathbf{NLC})$, then for every schedule S produced by agent B, $\langle M_2, t\rangle \in S \;\Rightarrow\; t \geq t_1$. The function $U_{\mathrm{est}}(\mathbf{E}, S, \mathbf{NLC})$ returns the estimated utility at the end of the episode if the agent follows schedule S and all non-local commitments in $\mathbf{NLC}$ are kept. Thus we may define the local scheduler as a function $\mathrm{LS}(\mathbf{E}, \mathbf{C}, \mathbf{NLC})$ returning a set of schedules

[2] Note that quality does not accrue after a task group's deadline.

[3] Other commitments, such as to the earliest start time of a task, may also prove useful.

$\mathbf{S} = \{S_1, S_2, \ldots, S_m\}$. More information about this interface between the local scheduler and the coordination component may be found in Section 5.7 and [Garvey *et al.*, 1994].

This is an extremely general definition of the local scheduler, and is the minimal one necessary for the GPGP coordination module. Stronger definitions than this will be needed for more predictable performance. Ideally, the optimal local scheduler would find both the schedule with maximum utility $\hat{S}_U$ and the schedule with maximum utility that violates no commitments $\hat{S}_{\bar{V}}$. In practice, however, a heuristic local scheduler will produce a set of schedules where the schedule of highest utility S_U is not necessarily optimal: $\mathrm{U}(\mathbf{E}, S_U, \mathbf{NLC}) \leq \mathrm{U}(\mathbf{E}, \hat{S}_U, \mathbf{NLC})$.

5.6 The Coordination Mechanisms

The role of the coordination mechanisms is to provide constraints to the local scheduler (by modifying portions of the subjective task structure of the episode $\mathbf{E}$ or by making commitments in $\mathbf{C}$) that allow the local scheduler to construct objectively better schedules. The modules fulfill this role by (variously) communicating private portions of its task structures to other agents, 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 or within portions controllable by *multiple* agents.[4]

The five mechanisms we will describe in this dissertation form a basic set that provides similar functionality to the original partial global planning algorithm as explained in [Decker and Lesser, 1992]. 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 one to update utilities across agents as discussed in the next section, or to balance the load better between agents. 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 as discussed in the next section. 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).

5.6.1 The Substrate Mechanisms

All the specific coordination mechanisms rest on a common substrate that handles information gathering actions (new task group arrivals and receiving communications), invoking the local scheduler and choosing a schedule to execute (including dealing with violated commitments), and deciding when to terminate. Information gathering is done at the start of problem solving, whenever the agent is otherwise idle (but not ready to terminate), and when communications are expected from other agents. Communications are expected in response to certain events (such as after the arrival of a new task group) or as indicated in the set of non-local commitments $\mathbf{NLC}$. This is the minimal general information gathering policy.[5]

[4] 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.

[5] The minimal policy would examine each element of $\mathbf{NLC}$ at the appointed time and if the local schedule had changed so that the reception of the information would no longer have any effect, the associated information gathering action could be skipped.

Termination of a task group occurs for an agent when the agent is idle on that task group, has no expected communications, and no outstanding commitments.

Choosing a schedule is more complicated. From the set of schedules $\mathbf{S}$ returned by the local scheduler, two particular schedules are identified: the schedule with the highest utility S_U and the best committed schedule S_C. If they are the same, then that schedule is chosen. Otherwise, we examine the sum of the changes in utility for each commitment. Each commitment, when created, is assigned the estimated utility U_{est} for the task group of which it is a part. This utility may be updated over time (when other agents depend on the commitment, for example). We then choose the schedule with the largest positive change in utility. This allows us to abandon commitments if doing so will result in higher overall utility. The coordination substrate does not use the local scheduler's utility estimate U_{est} directly on the entire schedule because it is based only on a local view, and the coordination mechanism may have received non-local information that places a higher utility on a commitment than it has locally. This method of choosing a schedule from a set is the one used by the GPGP approach, other solutions are also possible.

For example, at time t agent A may make a commitment C_1 on task $T \in \mathcal{T}_1 \in \mathbf{E}$ that results in a schedule S_1. C_1 initially acquires the estimated utility of the task group of which it is a part, $U(C_1) \leftarrow U_{\text{est}}(\{\mathcal{T}_1\}, S_1, B_A(\mathbf{NLC}))$. Let $U(C_1) = 50$. After communicating this commitment to agent B (making it part of $B_B(\mathbf{NLC})$, agent B uses the commitment to improve $U_{\text{est}}(\{\mathcal{T}_1\}, S_2, B_B(\mathbf{NLC}))$ to 100. A coordination mechanism can detect this discrepancy and communicate the utility increase back to agent A, so that when agent A considers discarding the commitment, the coordination substrate recognizes the non-local utility of the commitment is greater than the local utility.[6]

If both schedules have the same impact, the one that is more negotiable is chosen. Every commitment has a negotiability index (high, medium, or low) that indicates (heuristically) the difficulty in rescheduling if the commitment is broken. This index is set by the individual coordination mechanisms. For example, hard coordination relationships like enables that cannot be ignored will trigger commitments with low negotiability.

If the schedules are still equivalent, the shorter one is chosen, and if they are the same length, one is chosen at random.[7] After a schedule S is chosen, if $\text{Violated}(S)$ is not empty, then each commitment $C \in \text{Violated}(S)$ is replaced with its alternative $\mathbf{C} \leftarrow \mathbf{C} \setminus C \cup \text{Alt}(C, S)$. If the commitment was made to other agents, the other agents are also informed of the change in the commitment. While this could potentially cause cascading changes in the schedules of multiple agents, it generally does not for three reasons: first, as we mentioned in the previous paragraph less important commitments are broken first; secondly, the resiliancy of the local schedulers to solve problems in multiple ways tends to damp out these fluctuations; and third, agents are time cognizant resource-bounded reasoners that interleave execution and scheduling (i.e., the agents cannot spend all day arguing over scheduling details and still meet their deadlines). We have observed this useful phenomenon before [Decker and Lesser, 1993a] and plan to analyze it in future work.

[6] While it is clear that without this policy the system of agents will perform non-optimally, it is not clear how often the situation occurs or what the performance hit is. Future work will have to examine the costs and benefits of this policy; for this reason we do not include this mechanism among the five examined in this dissertation.

[7] It would be possible to allow the local scheduler to choose, as well.

5.6.2 Mechanism 1: Updating Non-Local Viewpoints

Remember that each agent has only a partial, subjective view of the current episode. Agents can, therefore, communicate the private portions of their subjective task structures to develop better, non-local, views of the current episode. They could even communicate all of their private structural information, in an attempt to develop a global subjective view. What non-local information to communicate and when has been an important area of study in both DAI (for example, the ten years of work on the DVMT and Carver's follow-on work called DRESUN [Carver *et al.*, 1991]) and distributed operating systems (for example, [Stankovic, 1985, Mirchandaney *et al.*, 1989]).

The GPGP mechanism described here can communicate no private information ('none' policy, no non-local view), or all of it ('all' policy, global view), or take an intermediate approach ('some' policy, partial view): an agent communicates to other agents only the private structures that are related by some coordination relationship to a structure known by the other agents. The process of detecting coordination relationships between private and shared parts of a task structure is in general very domain specific, so in the experiments presented later in the dissertation we model this process by a new information gathering action, *detect-coordination-relationships*, that takes some fixed amount of the agent's time. This action is chosen when a new task group arrives (which adds new information to the agent's private task structures). Updates can occur at different levels of detail in the task structure; this selective communication is similar to hierarchical communication discussed by von Martial and Montgomery [v. Martial, 1992, Durfee and Montgomery, 1990].

The set $\mathbf{P}$ of privately believed tasks or methods at an agent A (tasks believed at arrival time by A only) is then $\{x \mid \mathit{task}(x) \wedge \forall a \in \mathbf{A} \setminus A,\ \neg B_A(B_a^{\mathrm{Ar}(x)}(x))\}$, where $\mathbf{A}$ is the set of all agents and $\mathrm{Ar}(x)$ is the arrival time of x. Given this definition, the action *detect-coordination-relationships* returns the set of private coordination relationships $\mathbf{PCR} = \{r \mid T_1 \in \mathbf{P} \wedge T_2 \notin \mathbf{P} \wedge [r(T_1, T_2) \vee r(T_2, T_1)]\}$ between private and mutually believed tasks. The action does not return what the task T_2 is, just that a relationship exists between T_1 and some otherwise unknown task T_2. For example, in the DVMT, we have used the physical organization of agents to detect that Agent A's task T_1 in an overlapping sensor area is in fact related to some unknown task T_2 at agent B (i.e. $B_A(B_B(T_2))$) [Decker and Lesser, 1992, Decker *et al.*, 1991]. The non-local view coordination mechanism then communicates these coordination relationships, the private tasks, and their context: if $r(T_1, T_2) \in \mathbf{PCR}$ and $T_1 \in \mathbf{P}$ then r and T_1 will be communicated by agent A to the set of agents $\{a \mid B_A(B_a(T_2))\}$.

For example, Figure 5.3 shows the local subjective beliefs of agents A and B after the communication from one another due to this mechanism. The agents' initial local view was shown previously in Figure 5.1. In this example, T_3 and T_4 are two elements in Agent B's private set of tasks $\mathbf{P}$, $\mathsf{facilitates}(T_4, T_1, \phi_d, \phi_q) \in \mathbf{PCR}$ (the facilitation relates a private task to a mutually believed task), and $\mathsf{enables}(T_4, T_3)$ is completely local to Agent B (it relates two private tasks). At the start of this section we mentioned that coordination relationships exist between portions of the task structure controllable by different agents (i.e., in $\mathbf{PCR}$) and within portions controllable by multiple agents. We'll denote the complete set of coordination relationships as $\mathbf{CR}$; this includes all the elements of $\mathbf{PCR}$ and all the relationships between non-private tasks. Some relationships are entirely local—between private tasks—and are only of concern to the local scheduler. The purpose of this coordination mechanism is the exchange of information that expands the set of coordination relationships $\mathbf{CR}$. Without this mechanism

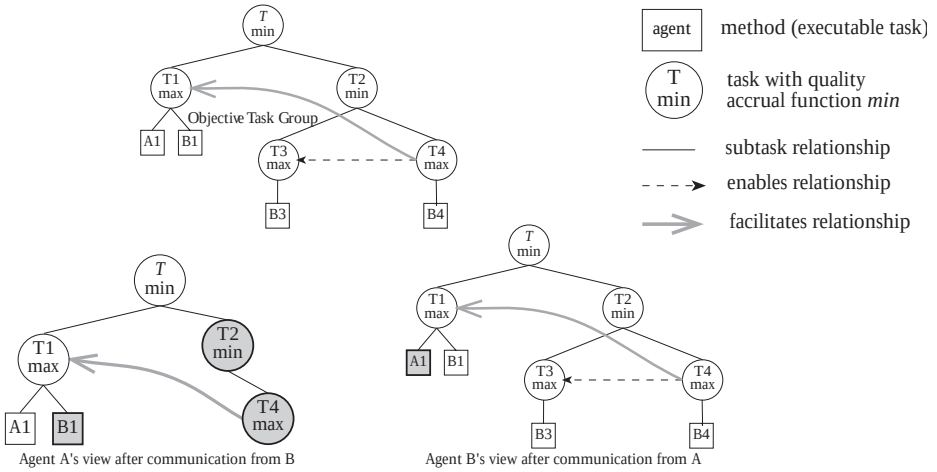

Figure 5.3. Agents A and B's local views after receiving non-local viewpoint communications via mechanism 1. The previous figure shows the agents' initial states.

in place, $\mathbf{CR}$ will consist of only non-private relationships, and none that are in $\mathbf{PCR}$. Since the primary focus of the coordination mechanisms is the creation of social commitments in response to coordination relationships (elements of $\mathbf{CR}$), this mechanism can have significant indirect benefits. In environments where $|\mathbf{PCR}|$ tends to be small, very expensive to compute, or not useful for making commitments (see the later sections), this mechanism can be omitted, or used selectively in a situation-dependent manner.

5.6.3 Mechanism 2: Communicating Results

The result communication coordination mechanism has three possible policies: communicate only the results necessary to satisfy commitments to other agents (the minimal policy); communicate this information plus the final results associated with a task group ('TG' policy), and communicate all results ('all' policy).[8] Extra result communications are broadcast to all agents, the minimal commitment-satisfying communications are sent only to those agents to whom the commitment was made (i.e., communicate the result of T to the set of agents $\{A \in \mathbf{A} \mid B(B_A(C(T))\}$.

5.6.4 Mechanism 3: Handling Simple Redundancy

Potential redundancy in the efforts of multiple agents can occur in several places in a task structure. Any task that uses a 'max' quality accumulation function (one possible semantics for an 'OR' node) indicates that, in the absence of other relationships, only one subtask needs to be done. When such subtasks are complex and involve many agents, the coordination of these agents to avoid redundant processing can also be complex; we will not address the general redundancy avoidance problem in this dissertation (see instead [Lesser, 1991]). In the original

[8] Communicating all results is necessary if there are no commitments but there are enabling relationships between tasks at different agents.

PGP algorithm and domain (distributed sensor interpretation), the primary form of potential redundancy was simple method redundancy—the same result could be derived from the data from any of a number of sensors. The coordination mechanism described here is meant to address this simpler form of potential redundancy.

The idea behind the simple redundancy coordination mechanism is that when more than one agent wants to execute a redundant method, one agent is randomly chosen to execute it and send the results to the other interested agents. This is a generalization of the 'static' organization algorithm discussed by Decker and Lesser [Decker and Lesser, 1993b]—it does not try to load balance, and uses one communication action (because in the general case the agents do not know beforehand, without communication, that certain methods are redundant[9]). The mechanism considers the set of potential redundancies $\mathbf{RCR} = \{r \in \mathbf{CR} \mid [r = \mathsf{subtask}(T, \mathbf{M}, \min)] \wedge [\forall M \in \mathbf{M}, \mathit{method}(M)]\}$. Then for all methods in the current schedule S at time t, if the method is potentially redundant then commit to it and send the commitment to $\mathrm{Others}(\mathbf{M})$ (non-local agents who also have a method in $\mathbf{M}$):

$$\begin{array}{rcl} [\langle M, t_M \rangle \in S] \ \wedge & & \\ [\mathsf{subtask}(T, \mathbf{M}, \min) \in \mathbf{RCR}] \ \wedge & & \\ [M \in \mathbf{M}] & \Rightarrow & [C(\mathsf{Do}(M, \mathrm{Q}_{\mathrm{est}}(M, \mathrm{D}(M), S))) \in C] \ \wedge \\ & & [\mathsf{comm}(M, \mathrm{Others}(\mathbf{M}), t) \in \mathcal{I}] \end{array}$$

See for example the top of figure 5.4—both agents commit to Do their methods for T_1.

After the commitment is made, the agent must refrain from executing the method in question if possible until any non-local commitments that were made simultaneously can arrive (the communication delay time δ). This mechanism then watches for multiple commitments in the redundant set $(\mathsf{subtask}(T, \mathbf{M}, \min) \in \mathbf{RCR}$, $M_1 \in \mathbf{M}$, $M_2 \in \mathbf{M}$, $C(\mathsf{Do}(M_1, q)) \in C$, and $B_B(C(\mathsf{Do}(M_2, q))) \in \mathbf{NLC}$[10] and if they appear, a unique agent is chosen randomly (but identically by all agents) from those with the best commitments to keep its commitment. All the other agents can retract their commitments. For example the bottom of figure 5.4 shows the situation after Agent B has retracted its commitment to Do B_1. If all agents follow the same algorithm, and communication channels are assumed to be reliable, then no second message (retraction) actually needs to be sent (because they all choose the same agent to do the redundant method). In the implementation described later, identical random choices are made by giving each method a unique random identifier, and then all agents choose the method with the 'smallest' identifier for execution.

Initially, all Do commitments initiated by the redundant coordination mechanism are marked highly negotiable. When a redundant commitment is discovered, the negotiability of the remaining commitment is lowered to medium to indicate the commitment is somewhat more important (i.e. less negotiable).

---

[9] The detection of redundant methods is domain-dependent, as discussed earlier. Since we are talking here about simple, direct redundancy (i.e. doing the exact same method at more than one agent) this detection is very straight-forward.

[10] Read "M_1 and M_2 are redundant, and I am doing M_1 and I know that B has committed to me to do M_2".

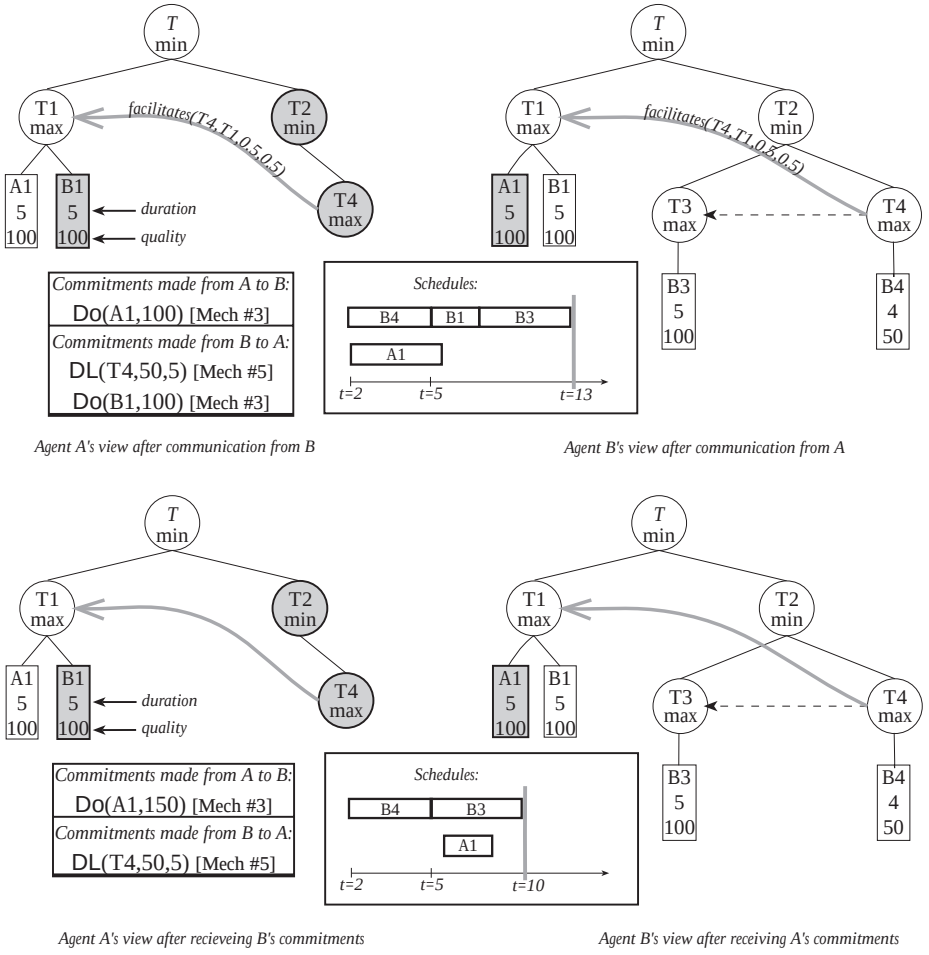

Figure 5.4. A continuation of the previous figure. At top: agents A and B propose certain commitments to one another via mechanisms 3 and 5. At bottom: after receiving the initial commitments, mechanism 3 removes agent B's redundant commitment.

5.6.5 Mechanism 4: Handling Hard Coordination Relationships

Hard coordination relationships include relationships like $\mathsf{enables}(M_1, M_2)$ that indicate that M_1 must be executed before M_2 in order to obtain quality for M_2. Like redundant methods, hard coordination relationships can be culled from the set $\mathbf{CR}$. The hard coordination mechanism further distinguishes the direction of the relationship—the current implementation only creates commitments on the predecessors of the $\mathsf{enables}$ relationship. We'll let $\mathbf{HPCR} \subset \mathbf{CR}$ indicate the set of potential hard predecessor coordination relationships. The hard coordination mechanism then looks for situations where the current schedule S at time t will produce quality for a predecessor in $\mathbf{HPCR}$, and commits to its execution by a certain deadline both locally and socially:

$$\begin{array}{rcl} [\mathrm{Q}_{\mathrm{est}}(T, \mathrm{D}(T), S) > 0] \ \wedge & & \\ [\mathsf{enables}(T, \mathbf{M}) \in \mathbf{HPCR}] & \Rightarrow & [C(\mathsf{DL}(T, \mathrm{Q}_{\mathrm{est}}(T, \mathrm{D}(T), S), t_{\mathrm{early}})) \in C] \ \wedge \\ & & [\mathsf{comm}(C, \mathrm{Others}(\mathbf{M}), t) \in \mathcal{I}] \end{array}$$

The next question is, by what time (t_{early} above) do we commit to providing the answer? One solution, usable with any local scheduler that fits our general description in Section 5.5, is to use the $\min t$ such that $\mathrm{Q}_{\mathrm{est}}(T, \mathrm{D}(T), S) > 0$. In our implementation, the local scheduler provides a query facility that allows us to propose a commitment to satisfy as 'early' as possible (thus allowing the agent on the other end of the relationship more slack). We take advantage of this ability in the hard coordination mechanism by adding the new commitment $C(\mathsf{DL}(T, \mathrm{Q}_{\mathrm{est}}(T, \mathrm{D}(T), S), \text{"early"}))$ to the local commitments C, and invoking the local scheduler $LS(\mathbf{E}, \mathbf{C}, \mathbf{NLC})$ to produce a new set of schedules $\mathbf{S}$. If the preferred, highest utility schedule $S_U \in \mathbf{S}$ has no violations (highly likely since the local scheduler can simply return the same schedule if no better one can be found), we replace the current schedule with it and use the new schedule, with a potentially earlier finish time for T, to provide a value for t_{early}. The new completed commitment is entered locally (with low negotiability) and sent to the subset of interested other agents.

If redundant commitments are made to the same task, the earliest commitment made by any agent is kept. If more than one commitment is for the same time, then we choose the agent committing to the highest quality. Any remaining ties are randomly broken by the same method as described in the previous section.

Currently, the hard coordination mechanism is a proactive mechanism, providing information that might be used by other agents to them, while not putting the individual agent to any extra effort. Other future coordination mechanisms might be added to the family that are reactive and request from other agents that certain tasks be done by certain times; this is quite different behavior that would need to be analyzed separately.

5.6.6 Mechanism 5: Handling Soft Coordination Relationships

Soft coordination relationships are handled analogously to hard coordination relationships except that they start out with high negotiability. In the current implementation the predecessor of a $\mathsf{facilitates}$ relationship is the only one that triggers commitments across agents, although $\mathsf{hinders}$ relationships are present. The positive relationship $\mathsf{facilitates}(M_1, M_2, \phi_d, \phi_q)$ indicates that executing M_1 before M_2 decreases the duration of M_2 by a 'power' factor related to ϕ_d

and increases the maximum quality possible by a 'power' factor related to ϕ_q (see [Decker and Lesser, 1993d] for the details). A more situation-specific version of this coordination mechanism might ignore relationships with very low 'power'. The relationship hinders$(M_1, M_2, \phi_d, \phi_q)$ is negative and indicates an *increase* in the duration of M_2 and a *decrease* in maximum possible quality. A coordination mechanism could be designed for hinders (and similar negative relationships) and added to the family. To be proactive like the existing mechanisms, a hinders mechanism would work from the *successors* of the relationship, try to schedule them late, and commit to an earliest start time on the successor. Figure 5.4 shows Agent B making a D commitment to do method B_4, which in turn allows Agent A to take advantage of the facilitates$(T_4, T1, 0.5, 0.5)$ relationship, causing method A_1 to take only half the time and produce 1.5 times the quality.

5.7 Interfacing the Coordination Mechanism with the Local Scheduler

In our work on real-time AI problem solving prior to this dissertation we have found that the interface between the decision-maker and the real-time scheduler needs to be complex and bidirectional. We have argued that this interface can usefully be modeled as a negotiation process [Garvey *et al.*, 1994]. In previous work on real-time scheduling, the interface between the scheduler and the application has been very simple. It is usually assumed that the application passes tasks on to the scheduler for scheduling and does not react when the scheduler is unable to schedule some tasks before their deadlines or only able to provide low quality solutions. A more complex interface is proposed in [Stankovic *et al.*, 1989] that allows the application to ask what-if questions of the scheduler and modulate the behavior of the scheduler, however these ideas have not been implemented to date. In our work on a complex, real-time, multi-agent problem-solver we have found that a more bidirectional, negotiation-based interface is useful. Such an interface should be useful in both multi-agent and complex, single-agent problem-solving environments where there are many ways to solve a problem and also multiple criteria with which to judge the potential solutions. I will briefly discuss the interface between higher-level decision-making and lower-level scheduling and acting, described as a negotiation process between the scheduler and the decision-maker. I will discuss why each part of the interface is useful, whether it can be implemented efficiently in a scheduler, and the effects of omitting that part.

As an example of what we mean by such an interface, consider a situation where multiple agents are working on a problem. Agent A has a method (Method A1) that enables the execution of an important method at another agent B. At this point the decision-maker at Agent A realizes that it should try to get the scheduler to schedule Method A1. It can do this by associating a *do commitment* with Method A1, meaning that it requests that the scheduler try to build schedules that execute Method A1. Agent A's scheduler returns a schedule that completes executing Method A1 at time 7. The decision-maker at Agent A tells other decision-makers that it can commit to giving them the result of Method A1 at time 8 (allowing time for communication to occur). At the second agent B, the decision-maker receives this message and passes it along to the scheduler, which reports back that time 8 is too late—the result is needed by time 6. Agent A's decision maker is informed of this feedback, and Agent A again invokes its scheduler, now with a *deadline commitment* to complete Method A1 by time 5 (to allow time for communication). Agent A's scheduler returns a schedule that commits to completing Method A1 by time 5 and Agent A communicates this information to the other agent, which

is now able to complete its method by the deadline. Meeting the tighter schedule might be possible only by making other tradeoffs—lowering the local quality produced or violating some other commitment. In this example, the decision-maker and scheduler at each agent are semi-autonomous subsystems, communicating bidirectionally and sometimes requiring more than one step to arrive at a satisfactory solution.

Given that significant communication in both directions is required in our layered approach, an obvious question is why have separate subsystems. There are many reasons why a distinct separation should exist between such a scheduler and decision-maker, including at least modularity, efficiency and reusability. Modularity suggests that separate functionality should be kept in separate modules with clearly defined interfaces. In general it is difficult for all current problem solving criteria to be encapsulated into an evaluation function and transmitted to the scheduler, because deciding what to do is a evolving computational *process*. From the scheduler's perspective, transmitting *all* potentially useful information about a schedule is also difficult and inefficient. Another reason why a separation should exist is that the subsystems work at different levels of abstraction. One of the roles of the decision maker is to constrain the search done by the scheduler, for example, by using commitments to tell the scheduler what parts of the task structure to focus on. While it is possible for the scheduler to use all available information to make such decisions itself, for efficiency reasons it is useful to have the decision maker constrain the search space for the scheduler. Another reason for separating scheduling and decision-making is that scheduling is a more generic activity and it should be possible to reuse schedulers in multiple applications. It is undesirable to reproduce a scheduler each time a new problem area is investigated.

I will discuss the scheduler/decision maker interface using examples from two different points-of-view: multi-agent environments such as those that have been addressed in this dissertation, and hard real-time environments that might be addressed in the future. The two points-of-view are not necessarily inconsistent with one another.

Multi-agent scenarios.

In the multiagent scenarios I discussed in Section 5.5, each method is executable by exactly one agent, however several agents may have identical methods for achieving quality for the same task. The goal of the coordinated decision-makers is to work together to produce the highest possible quality for as many task groups as possible, i.e., each attempts to maximize the *global* utility measure $\mathrm{U}(\mathbf{E}) = \sum_{\mathcal{T} \in \mathbf{E}} Q(\mathcal{T}, \mathrm{D}(\mathcal{T}))$. This is not straightforward, because each agent sees only some part of the total task structure, and it may be the case that no agent sees the entire structure. Thus a decision maker cannot simply ask the scheduler to optimize this global criteria. One kind of information that agents can communicate to one another is information about the task structures that they see (i.e., GPGP mechanism 1: updating non-local viewpoints). The decision-makers are responsible for coordinating their activity so as to avoid redundant method execution and allow relationships that extend across agents to be exploited or avoided as appropriate using the other GPGP coordination mechanisms. The role of the scheduler is to schedule execution of local methods according to criteria provided by the decision-maker.

Real-time scenarios.

In the real-time scenarios that we are working with, task groups arrive continuously, and require the use of multiple reusable physical resources (motorized tables, robot arms) and

consumable resources. There are often not enough resources to complete all task groups, but the decision-maker still attempts to maximize its total utility. Each task group may have a different maximum payoff $I(\mathcal{T})$, and the decision-maker tries to maximize $\mathrm{U}(\mathbf{E}) = \sum_{\mathcal{T} \in \mathbf{E}} I(\mathcal{T}) Q(\mathcal{T}, \mathrm{D}(\mathcal{T}))$. Here the criteria are known but the best mix of problems to solve is not. The role of the scheduler here is to *endorse* the execution of time-critical control codes on multiple hardware platforms, using shared, private, and consumable resources. Some task groups, representing periodic, maintenance, or operating system activities, will be constantly present and their execution will be absolutely guaranteed.

5.7.1 Scheduler Inputs

The decision-maker *proposes* to the scheduler that a solution to each newly arriving task group be added to the schedule of methods to be executed. A basic request to the scheduler (its input) consists of four things: the task structures to be scheduled, a set of commitments, a set of non-local commitments, and a runtime indication.

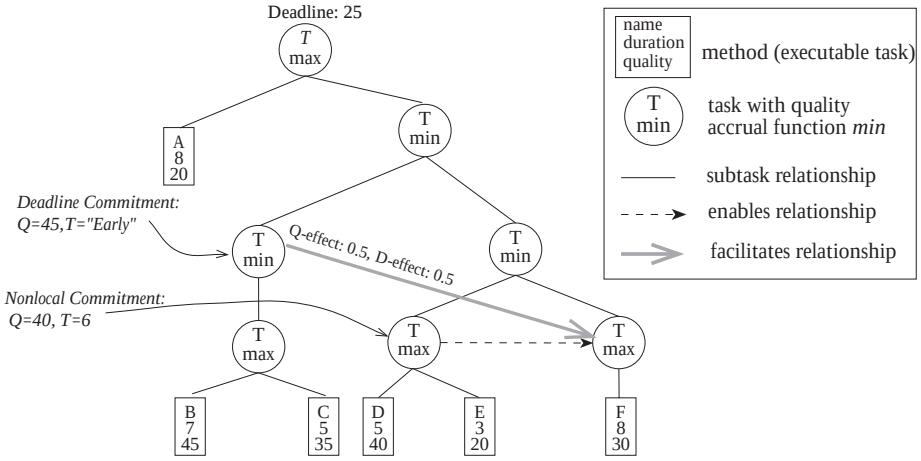

Figure 5.5. An example of a complete input specification to the scheduler.

The task structures to be scheduled $\mathbf{E}_S$, should include some indication of what aspects of those structures have changed since the scheduler was last invoked. If we write $B_A^t(X)$ to indicate what agent A believes at time t about X, then the scheduler at agent A at time t has access to $B_A^t(\mathbf{E})$, and $B_A^t(\mathbf{E}) \setminus B_A^{t-1}(\mathbf{E})$.

The set of commitments $\mathbf{C}$ are constraints that the invoker would like the scheduler to try to satisfy. We have defined three types of commitments:

- $C(\mathsf{Do}(T, q))$ is a commitment to 'do' (achieve quality for) T and is satisfied at the time t when $Q(T, t) \geq q$. A 'don't' commitment is also possible.
- $C(\mathsf{DL}(T, q, t_{dl}))$ is a 'deadline' commitment to do T by time t_{dl} and is satisfied at the time t when $[Q(T, t) \geq q] \wedge [t \leq t_{dl}]$. A $C(\mathsf{Do}(T, q))$ is really shorthand for $C(\mathsf{DL}(T, q, \mathrm{D}(T)))$.

- $C(\mathsf{EST}(T, q, t_{est}))$ is a 'earliest start time' commitment[11] to not begin work on T before time t_{est} and is satisfied at the time t_{est} iff $\forall t \leq t_{est},\ Q(T, t) \leq q$.

The importance of local commitments such as these are as *soft* constraints on the possible solutions. Any scheduler that can schedule real-time method executions can already deal with hard constraints such as deadlines and earliest start times. Hard commitments can be used to provide *guarantees* [Cheng *et al.*, 1988] by requiring commitments to be satisfied in all valid schedules.

Soft commitments are needed to handle the coordination of multiple agents where there is more than one way to solve a task or where there are soft coordination relationships such as facilitates. They are also useful in real-time systems, as shown by Spring's use of *endorsements* [Cheng *et al.*, 1988] to indicate commitments that may only be violated when more important tasks arrive. When invoking the scheduler in a query mode, the decision-maker may also supply the symbolic values 'early' for a deadline commitment and 'late' for an earliest start time commitment, which indicates to the scheduler that it should attempt to satisfy the commitment as early or late as possible.

If a scheduler does not provide the ability to specify soft commitments, it is possible in some situations for the decision-maker to achieve the same results by repeated execution of scheduler queries using hard commitments (even changing the task structure, if need be). We believe it will always be more efficient to add the ability to interpret soft constraints to the scheduler than to play guessing games by invoking the scheduler multiple times.

The set of non-local commitments **NLC** are commitments that the scheduler can assume will be satisfied. These are of the form of the commitments mentioned above and tell the scheduler to expect to receive the indicated results at the indicated time. In general, the scheduler would not concern itself with the reliability of this information, which is would be determined by the decision-maker. A sophisticated scheduler might concern itself with the reliability of these non-local commitments, but most schedulers assume this information is reliable.

In multi-agent problems, non-local commitments can be used to communicate work that will be done by other agents. This component is necessary for achieving coordinated behavior in complex domains. These non-local commitments might be created at run time by the decision-makers, or they might be derived from pre-defined 'social laws' [Shoham and Tennenholtz, 1992] that all agents agree to, or are constructed to, satisfy. The most important component of an agent's particular organizational role is the set of non-local commitments it forces the agent to adhere to. Another effect of NLCs in multi-agent problems is the triggering of non-local effects (coordination relationships); each non-local deadline commitment, for example, implies an earliest start time on the 'affected' end of any relationships. For hard relationships like enables this implies a hard earliest start time; for soft relationships it actually expands the search space (since each affected task can be started either before or after the earliest start time with different results).

In real-time problems as well, non-local commitments are important. In our example domain not all of the hardware is under the control of the real-time operating system scheduler,

[11] In fact this is more general than a standard earliest start time constraint, in that it allows some nonzero amount of work to be done on T as long as quality does not go above the threshold. Standard earliest start times can be modeled with a q value of 0.

in particular the robot arms run on separate hardware with a separate specialized execution controller. The only way for the RT scheduler to function is to allow the robot hardware to make non-local commitments (in this case, worse-case execution time guarantees) about certain physical activities that are not directly under the control of the real-time scheduler.

Another way to look at non-local commitments is that they give the decision-maker a way to only partially flesh out a task structure and to direct a scheduler's search, potentially making it more efficient. NLCs on elements of MAX (OR) nodes provide initial lower bound 'best' estimates, while NLCs on MIN (AND) elements provide upper bounds. Both types can be used to prune search in time-constrained situations—NLC's below MAX (OR) nodes could prune all their siblings from local consideration; NLCs below MIN (AND) nodes can only locally prune below the node with the NLC. For example, if the task 'get food for the party' consists of the subtasks 'buy prepared food' OR 'make food from leftovers', then if another agent commits to 'buy prepared food' I can prune this entire substructure from my schedule. On the other hand, if 'get food for the party' consists of the subtasks 'buy ingredients' AND 'make food from ingredients', then a non-local commitment to 'buy ingredients' prunes that subtask (and any below it) only; I still need to schedule to 'make food from ingredients'.

Another potential use for non-local commitments is to allow the decision-maker to direct the search of the scheduler. The decision-maker can use non-local commitments to ask questions such as, assuming quality is achieved in this part of the task structure, how could we take advantage of that in other parts of the task structure. This could be useful in situations where the cost of the information gathering associated with expanding a task structure is potentially large. It could also be used in situations where the scheduler has successfully produced a schedule to satisfy one part of a task structure and the decision-maker now wants to focus the schedulers attention on another part that can begin execution when the previously scheduled work is completed. In time-constrained situations such non-local commitments can reduce search for the scheduler by allowing it to prune committed portions of the task structure.

Various mechanisms for controlling the runtime of the scheduler can include: a hard deadline by which the scheduler should complete; a satisficing value for a schedule (the scheduler completes when a schedule of at least this value is found); or a decision-theoretic tradeoff function that indicates the added value of spending time finding better schedules versus executing the first element of the current schedule[Russell and Wefald, 1991]. In non-real-time scenarios, this might not be particularly important as long as the runtime of the scheduler is small compared to the grain size of application tasks. In real-time scenarios it is crucial to at least be able to predict the worst-case performance of the scheduler.

5.7.2 Scheduler Output

The output from the scheduler after an invocation should include at least one valid schedule, a list of satisfied commitments, a list of violated commitments with alternatives, an indication of tasks that should be scheduled but are not, and an indication of the value of each returned schedule with respect to some fixed set of criteria.

A set of valid schedules $\mathbf{S}$ is returned that do a satisfactory job of satisfying the problem given to the scheduler. An individual schedule $S \in \mathbf{S}$ consists of at least a set of methods and start times: $S = \{\langle M_1, t_1\rangle, \langle M_2, t_2\rangle, \ldots, \langle M_n, t_n\rangle\}$. This output is of course necessary,

and forms the initial *proposal* in the negotiation process. The remaining items provide an explanation of this proposal.

The next three items returned (satisfied commitments, violated commitments with alternatives, and multi-criteria schedule values) are not *necessary* for the scheduler to provide, because they can all be derived mathematically from the schedule itself and the set of non-local commitments. However, for practical implementations, the scheduler often has this information at hand, or can collect it during schedule generation, and it would be expensive to recompute.

The set of input commitments that are satisfied in a schedule $\mathrm{Satisfied}(S)$ is returned ($\forall S \in \mathbf{S}, \mathrm{Satisfied}(S) \subset \mathbf{C}$). If the scheduler supports symbolic local commitments like 'early' deadline commitments and 'late' earliest start time, then it must also supply an indication of when the commitment is expected to be satisfied in the schedule $\mathrm{SatTime}(C, S)$. For example, if $C_1 = \mathsf{DL}(T, q, \text{'early'})$ and $C_1 \in \mathrm{Satisfied}(S)$ then $\mathrm{SatTime}(C, S) = \min t \leq \mathrm{D}(T)$ s.t. $\mathrm{Q}_{\mathrm{est}}(T, t, S) \geq q$.

The set of input commitments that are violated in a schedule $\mathrm{Violated}(S)$ is returned. For each violated commitment, a proposed modification to the commitment that the scheduler *is* able to satisfy ($\mathrm{Alt}(C, S)$) is also returned. For earliest start time and deadline commitments this involves a proposed new time and/or minimum quality. For do/don't commitments this involves a recommended retraction or a reduced minimum quality value. For example, for a violated deadline commitment $C(\mathsf{DL}(T, q, t_{dl})) \in \mathrm{Violated}(S)$ the function $\mathrm{Alt}(C, S)$ returns an alternative commitment $C(\mathsf{DL}(T, q, t^{\star}_{dl}))$ where $t^{\star}_{dl} = \min t$ such that $Q(T, t) \geq q$ if such a t exists, or NIL otherwise.

The knowledge that certain commitments are satisfied or violated is absolutely necessary to the decision-maker that uses commitments, regardless of the domain.

An indication of the "value" of the schedules that were returned according to several objective criteria. Some of the objective functions that can be measured include the total quality for all scheduled task groups,[12] the number of task groups that do/do not complete before their deadline, the amount of slack time available in the schedule to allow easy scheduling of new tasks and/or allow time for tasks to take longer than expected to run, and the number (or weighted value) of the commitments that are not satisfied in the schedule (our decision criteria was described in Section 5.6.1).

Complex real problems invariably involve multiple evaluation criteria that must be balanced with one another; we view this balancing as the role of the decision-maker, and the scheduler attempts to maximize the current criteria, often returning multiple schedules (e.g., one that best satisfies each of the current criteria.) While the ability to evaluate a schedule with respect to certain criteria could be implemented outside of the scheduler, the ability to attempt to optimize certain criteria can only be placed in the scheduler.

In our multi-agent system, for example, Agent A's decision-maker might make a commitment to $\mathsf{Do}(M_1, q)$, and tell other agents. As time passes, Agent A will need to make a decision about whether to abandon the commitment—perhaps the scheduler finds a 'better' schedule that bypasses the execution of M_1. The scheduler cannot make this decision because it can only evaluate the schedule via local criteria. If another agent B uses the result of M_1 to produce a very high quality solution that A could not produce alone, B's decision-maker can notify A's

[12] This could be a weighted sum if task group importance varies, or some more complex function if desired.

decision-maker about the updated utility of the commitment. The decision-maker is now in a much better position to decide whether to abandon the local commitment to M_1 than the scheduler is, and the process did not involve the agent's exchanging all of their knowledge and all of their current schedules.

A real-time example of changing criteria can occur when time pressure is not constant. Under normal processing, the scheduler may process all task groups, attempting to produce maximal quality. However, when feedback from the scheduler indicates that it is unable to successfully schedule all task groups, the decision-maker may switch to a 'time-pressured' mode where the scheduler is directed to only produce minimum quality for every task. Such 'mode-changing' behavior can be directed by a decision-maker only by using feedback from the scheduler. Other work has shown such mode-changing behaviors as pre-compiled into the scheduler for efficiency purposes[Hayes-Roth *et al.*, 1988].

A minimal list of tasks in the task structure that the schedule is not providing quality for but would need to have quality to allow their task group to achieve non-zero quality. Such tasks can result from the scheduler not having any local methods to generate quality for the tasks (either because the task structure is distributed across agents and those tasks have methods at some other agent(s), or because the agent has not yet done the information gathering necessary to determine what methods are available for the task.) Such tasks can also result from the scheduler not being able to schedule the execution of all methods known to it because of deadlines or other constraints.

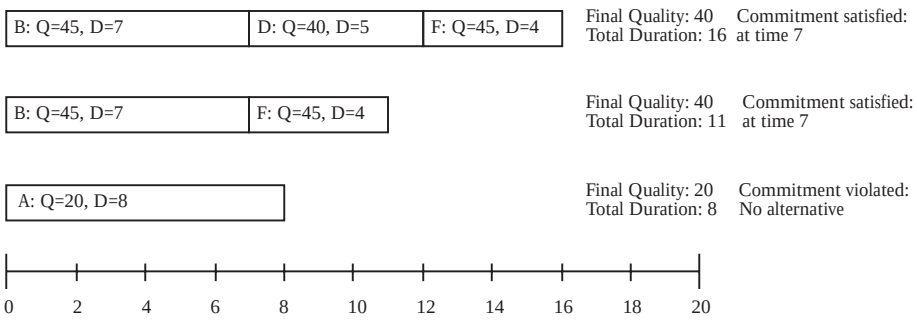

Figure 5.6. An example of the output of the scheduler for the example problem given above.

A summary of the output of the scheduler for the example problem given above is shown in Figure 5.6. In this example three schedules are returned. The bottom one is the schedule generated by the *minimum duration* generator and produces a fast, low quality result, violating the given deadline commitment because no quality is ever generated for the committed task. The middle schedule is generated by the *highest quality* generator and produces the highest possible quality in the fastest possible time, satisfying the deadline commitment at time 7. The top schedule is generated by the *minimum nonlocal reliance* generator and produces the highest quality possible completely locally, not relying on the given nonlocal commitment, also satisfying the deadline commitment at time 7. Which of these schedules is chosen by the decision-maker depends on the current evaluation criteria. If the fastest possible, acceptable

result is desired, perhaps because of a large workload of other tasks, then the bottom schedule is chosen. If the best possible result in the minimum possible time is desired, the the middle schedule is chosen. If the best possible result that does not rely on other agents is desired (possibly because of other work that those agents need to do or a concern about the other agent's reliability) then the top schedule is chosen.

5.7.3 Interfacing the Coordination Mechanism with the Local Scheduler: Discussion

We will first return to the main reasons for a bidirectional interface separating scheduling from decision making—reusability, modularity, and efficiency.

Of the various reasons for developing a complex bidirectional interface, the reusability argument is the most clear. A scheduling component that attempts to find optimal method execution times for arbitrary task structures and evaluation criteria is useful in many domains, regardless of whether they have real-time or distributed problem solving components. A more important related question is whether the extra capabilities required by this interface can be provided cheaply by an otherwise efficient, reusable scheduler. Examining the input characteristics, we find that the input task structures are not much more than the specification of the problem to be solved, and that handling these constraints would not be an additional burden to a standalone scheduler. Commitments, amounting to preferred (soft) deadlines and earliest start times are also part of standard real-time scheduling specifications. So too are the mechanisms for controlling the runtime of the scheduler, for any scheduler that can schedule real-time tasks with deadlines. The only potentially unique feature is the input of non-local commitments. If non-local commitments are always taken at face value (no lies) then they can be used for making more efficient searches, as discussed earlier.

Examining the reusable scheduler's output characteristics for efficient implementation, we find the first two—the schedules themselves and which commitments are satisfied when—to be non-controversial parts of almost any real-time scheduler. So too is the list of violated commitments, but perhaps not the generated alternatives. Generating alternatives to violated commitments on executable methods is trivial—just look in the generated schedule and return the actual execution if found, or suggest retraction of the commitment otherwise. Generating alternatives for high-level task commitments may be somewhat more complex, depending on the internal structure of the scheduler and what information is efficiently available. As we mentioned earlier, this output characteristic was assigned to the scheduler because it is *usually* more efficient to compute there, but it can be (inefficiently) computed by the decision maker from the schedule and non-local commitments themselves if necessary. The same thing is true of schedule evaluations under multiple criteria—it is our experience that the scheduler can more efficiently calculate these evaluations that the decision maker. The scheduler usually has already computed these evaluations as part of its search process. The most unique and potentially expensive output characteristic is the production of a task list for which the scheduler desires quality but cannot produce. This behavior enables several sophisticated responses in multi-agent systems, such as contracting behavior on the part of the decision maker on behalf of the scheduler, but is not part of the standard definition of scheduling problems and undoubtedly causes extra overhead. We plan to analyze how this behavior can be efficiently provided and under what circumstances it is useful in future work.

The other two reasons for a complex bidirectional interface, efficiency and modularity, are closely tied. Efficiency is primarily the ability to search on, or attempt to optimize, specific

criteria effectively. Defining a carefully delimited scheduling problem, and even limited search criteria, allows for the construction of a efficient scheduler. In fact, multiple schedulers might be constructed, each optimized to search (perhaps in parallel) under a different criteria (best quality, least violated commitments, earliest finish time). This leads directly to the modularity argument—that in general it is difficult to encapsulate all problem solving criteria into a single evaluation function, and deciding on the correct criteria is an evolving computational process (to be handled by a separate decision maker, we argue).

5.8 Summary

This chapter discussed my third contribution: the development of the Generalized Partial Global Planning *family* of coordination mechanisms. The GPGP algorithm family 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 (answered by Mechanism 1, update non-local views); how and when to exchange the partial results of problem solving (answered by Mechanism 2, communicating results); how and when to make and break *commitments* to other agents about what results will be available and when (Mechanisms 3, 4, and 5, handling simple redundancy and hard and soft relationships). Each agent also has a heuristic local scheduler that decides what the agent should do next, based on its current view of the problem solving situation (including the commitments it has made), and a utility function. The coordination mechanism supplies 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).

GPGP attempts to extend PGP in several ways. First, it permits more agent heterogeneity and, when possible, more scheduling autonomy. Agents are still considered to be cooperative (working with a shared global utility criteria), but GPGP permits agents to have different local views (letting the scheduler work with *partially global* plans (really schedules). In the original PGP algorithm agents generally communicate complete schedules (at a single *a priori* fixed level of abstraction), whereas GPGP agents only communicate scheduling commitments to particular actions (at any dynamically decided level of abstraction). Agents can be heterogeneous in their capabilities as well—some agents can be faster at certain tasks or have unique capabilities (methods). However, GPGP is a family of coordination mechanisms and does not address problems of heterogeneous systems such as created shared languages for communication. GPGP is useful in cooperative distributed problem solving (CDPS) situations, but the current set of mechanisms would be vulnerable to lying in a non-cooperative multi-agent situation.

Secondly, GPGP extends PGP to dynamic environments and problem solving architectures where planning (scheduling) at a single level of detail is not practical. This extension includes communicating meta-level information at varying levels of detail and reacting appropriately to situations where there is more than one way to accomplish a high level task.

The third GPGP extension is to real-time problem solving, with the addition of hard deadlines and more flexible temporal commitments. However, the algorithm is only 'real-time' in the same sense as the underlying local scheduler (developed by Garvey [Garvey and Lesser, 1994]), and not in the strong sense of guaranteed computation like the Spring scheduler

[Stankovic *et al.*, 1989] (how these all can work together is a future research direction, see also the discussion in Section 5.7).

Because GPGP is a cooperative, team-oriented mechanism, it assumes that no agent has a consistently better view of problem solving (either because of the problem-solving algorithm or by some imposition of the environment). Our approach views the coordination mechanism as modulating local control, not supplanting it—a two-level process that makes clear the distinction between coordination behavior and local scheduling, and different from past and present Durfee work. It is also different from Corkill's work (which also had distinct local control [Corkill and Lesser, 1983]) in that it is dynamic in nature. This separation also allows GPGP to take advantage of recent advances in the planning and scheduling of computations.

GPGP mechanisms are reactions to the presence and quantitative features of coordination relationships. Coordination relationships, as they arise from the general task environment model, have domain-independent definitions. These relationships have quantitative properties such as how likely they are to appear, how difficult they are to detect, and how significant their effect is. A coordination algorithm defined in terms of detecting the existence and properties of CRs, and producing scheduling constraints, is more general than one written for a specific application such as the DVMT. Both domain-independence and modularity will allow for researchers and application designers to share ideas more easily.

GPGP is a family of algorithms—each of the five mechanisms described here are parameterized independently (the first two have three possible settings and the last three can be in or out) for a total of 72 combinations. For example, one specific family member that we will use experimentally in the next chapter is named the "balanced" algorithm. The "balanced" algorithm has Mechanisms 3, 4, and 5 'on', updates 'some' non-local views (Mechanism 1), and communicates commitments and finished task groups (Mechanism 2). Many of these combinations do not show significantly different performance in randomly generated episodes, as will be discussed in the next chapter, although they will allow for fine-tuning in specific applications. More mechanisms can (and have) been added to expand the family, but the family can also be enlarged by making each mechanism more situation-specific. For example, mechanisms can have their parameters set by a mapping from dynamic meta-level measurements such as an agent's load or the amount of real-time pressure. Mechanisms can be 'in' or 'out' for individual *classes* of task groups, or tasks, or even specific coordination relationships, that re-occur in particular environments. The cross product of these dynamic environmental cues provides a large but easily enumerated space of potential coordination responses that are amenable to the adaptation of the coordination mechanisms over time by standard machine learning techniques or case-based reasoning approaches. Such extensions fall under the heading of Future Work (Chapter 7). In the experimental section of dissertation (Chapter 6) we will only consider the coarsest parameterization of the mechanisms.

In the next chapter we will evaluate and show the performance of these mechanisms, using several sets of experiments that deal with performance of the GPGP mechanisms in different environments, using a real implementation (running in Common Lisp on multiple platforms, and being used by more than one person). The implementation uses as submodules a real implementation of Garvey's Design-To-Time real-time scheduler [Garvey and Lesser, 1994].

CHAPTER 6

EXPERIMENTS IN GENERALIZED PARTIAL GLOBAL PLANNING

> *"You doubt what I wrote? Let me show you." We now understand that what the Professor is asking us to watch is related to the figure [in his scientific paper]. We thus realize where this figure comes from. It has been* extracted *from the instruments of this room,* cleaned, redrawn, *and* displayed. *We also realize, however, that the images that were the last layer in the text, are the* end result *of a long process in the laboratory that we are now starting to observe.*
>
> *We are no longer asked to believe the text we read in 'Nature'; we are now asked to believe* our own eyes *. . . Do we see more or less than before? We can see more, since we have before our eyes not only the image but what the image is made of. On the other hand we see* less *because now each of the elements that makes up the final graph could be modified so as to produce a different visual outcome. . . . The Professor, for instance, is swearing at the gut saying it is a 'bad gut'.*
>
> *A guinea pig is placed on a table, under surgical floodlights, then anaesthetised, crucified, and sliced open. . . . Suddenly, we are much further from the paper world of the article. We are now in a puddle of blood and viscera, slightly nauseated by the extraction of the ileum from this little furry creature. In the last chapter, we admired the rhetorical abilities of the Professor as an author. Now, we realize that many other abilities are required in order to write a convincing paper later on. The guinea pig alone would not have been able to tell us anything . . . it was not mobilisable into a text and would not help to convince us. Only a part of its gut, tied up in the glass chamber and hooked up to a physiograph, can be mobilised in the text and add to our conviction. Thus, the Professor's art of convincing his readers must extend beyond the paper to preparing the ileum, to calibrating the peaks, to tuning the physiograph.*
>
> *— Bruno Latour, Science in Action*

These experiments involve a complete implementation of GPGP and a separately developed real-time local scheduler [Garvey and Lesser, 1993, Garvey *et al.*, 1994]. We will show how to decide when a particular mechanism (and thus GPGP) is useful, how some family members perform relative to a centralized algorithm, and what the space of possible coordination algorithms looks like for the five mechanisms currently defined. We analyze the performance of this family of algorithms through simulation in conjunction with the heuristic real-time local scheduler and randomly generated abstract task environments.

In these environments, the agents attempt to maximize the system-wide total utility (a quantity called 'quality', described later) by executing sequences of interrelated 'methods'. The agents do not initially have a complete view of the problem solving situation, and the execution of a method at one agent can either positively or negatively affect the execution of other methods

at other agents. We will show examples of the effect of the environment on the performance of a GPGP family member, and show an environment where family member A is better than B, and a different environment where B is better than A. We will return to the demonstration of meta-level information being more useful when there is a large amount of variance between episodes in an environment.

In this final section we will discuss experiments we have conducted with our implementation of these ideas:

- How should we decide when the addition of a particular mechanism is warranted?
- What is the performance of a system using all the mechanisms compared to a system that only broadcasts results? Compared to a system with a centralized scheduler?
- What is the performance space of the GPGP family, as delineated by the five existing mechanisms?

As we have stated several times in this dissertation, we do not believe that any of the mechanisms that collectively form the GPGP family of coordination algorithms are indispensable. What we can do is evaluate the mechanisms on the terms of their costs and benefits to cooperative problem solving both analytically and experimentally. This analysis and experimentation takes place with respect to a very general task environment that does not correspond to a particular domain. Doing this produces general results, but weaker than would be possible to derive in a single fixed domain because the performance variance between problem episodes will be far greater than the performance variance of the different algorithms within a single episode. Still, this allows us to determine broad characteristics of the algorithm family that can be used to reduce the search for a particular set of mechanism parameters for a particular domain (with or without machine learning techniques). We will also discuss statistical techniques (e.g. paired-response) to deal with the large between-episode variances that occur when using randomly-generated problems.

The first section of this chapter describes a set of experiments using a very simple simulation and task structure that looks only at the effect of the facilitates relationship on agent performance—in particular, the effect of the *power* of the relationship and the *likelihood* of its existence. The rest of the chapter is devoted to an exploration of the performance of the GPGP family of coordination mechanisms as described in the previous chapter.

6.1 Initial Experiments: The Effect of Facilitation Power and Likelihood

The simulation used in this first chapter section is driven by the environment, task, and agent characteristics we discussed in Chapter 3; the tasks represent abstract computations and their structure was a simple one-level-deep AND hierarchy. Experiments using the full TÆMS framework will be discussed in the rest of the chapter, beginning in Section 6.2. In the experiments below, there were two abstract task groups. One group of tasks had a mean time between subtask arrivals of 40% less than the other. Each task arrives uniformly randomly at an agent. Facilitation relationships are generated between tasks with a base probability that decreases linearly with the difference between task arrivals[1]. For example, if there are

[1]The reader should not be worried about 'linear' vs. 'exponential' here. An experiment not reported here showed that in an environment where the probability of a facilitates relationship drops off exponentially instead of linearly, the system response characteristics are similar to a linear environment with the *same number of detected relationships*.

642 tasks generated, and a base probability of 0.5, interrelated tasks are grouped into clusters approximately distributed as in the histogram in Figure 6.1. If A facilitates B and C, and C facilitates D, that cluster is of size 4. This gives an indication of the webs of commitment that may potentially exist. The total number of ways to distribute k tasks to n agents is n^k. The number of ways to distribute k tasks to i agents where each agent gets at least 1 task (surjections) is $i!S(k,i)$, where $S(k,i)$ are the Stirling numbers of the second kind. So the expected number of n total agents that are involved in a k-cluster is:

$$\sum_{i=1}^{n} i \frac{\binom{n}{i} i! S(k,i)}{n^k}$$

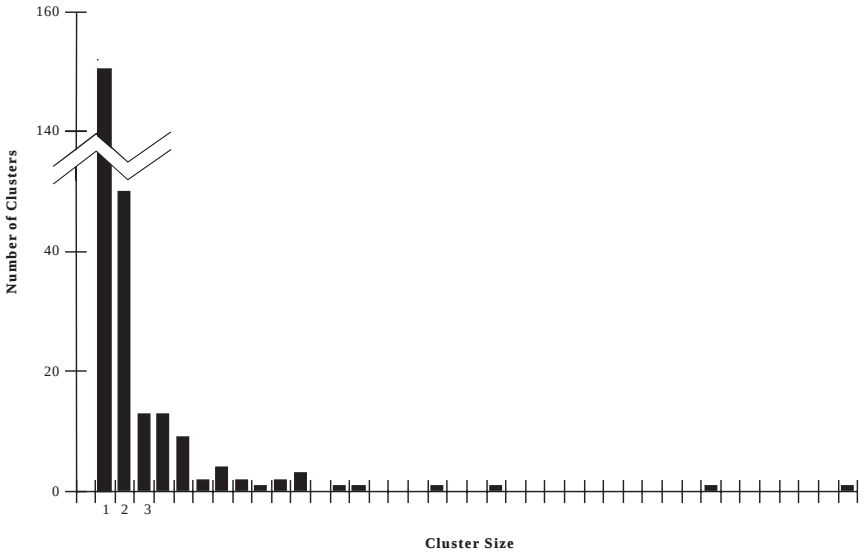

Figure 6.1. Histogram of sizes of task clusters for $\mathrm{Pr}_{\mathbf{fac}} = 0.5$, $N = 642$.

Each agent uses an early-version "design-to-time" (DTT) real-time local scheduler based on the concept of approximate processing[Decker *et al.*, 1990, Garvey and Lesser, 1993]. The DTT scheduler will choose a method for a task based on the amount of time available for that task and the other tasks currently on the agenda. The DTT scheduler may change the method being used during execution at a task monitoring point; in the experiments described here 50% of the work done before changing methods is lost. The DTT scheduler is boundedly rational for deadline constraints, and was modified to be boundedly rational for delay constraints (delaying the start of a task until the result of another task is received). Each class of tasks had 5 methods of varying quality and duration; each task execution was monitored by the DTT scheduler three times. The actual duration and quality values for each method for each task are randomly generated from normal distributions with means equal to the estimated values and variances as specified for the method/task combination. The shared global utility function by which agents are judged is the total quality of their individual task solutions. The design-to-time scheduling

algorithm used by each agent ensures that under normal circumstances (a required utilization (system load) of less than 3) less than 2% of the tasks ever miss a deadline. This is a property of the existence of low cost/low quality approximations.

Observations in the experiments that follow are made from average responses over 5 statistically generated runs of 1000 simulated world time units. The number of tasks that are actually generated depends on their arrival rate, which was varied. We also varied the *a priori* likelihood ($\mathrm{Pr}_{\mathrm{fac}}$) of a facilitates relationship between two tasks of the same class and the significance of the effect it has on the time of the *facilitated* tasks (called the *power* of the CR and measured by the percent reduction in time facilitated).

Each run consists of two sets of 4 agents. Each of the two agent-sets receives exactly the same set of tasks at exactly the same times—one agent-set uses the original DTT scheduling algorithm and no communication, the other agent-set uses the DTT scheduling algorithm modified to be boundedly rational with respect to *delay* constraints and to *always* detect and communicate coordination relationships. Each of the four agents in each agent-set receives precisely the same set of tasks as its counterpart in the other agent-set. The coordinating agents calculate delays as described in Figure 6.2. If a commitment fails, the coordinating agents simply break the failed commitment.

The primary performance metric in the experiments presented here is the percentage increase in quality between each pair of agents that received the same task set (a paired response), and then averaged across the agent pairs. This metric is indicated in the figures by APQI (average percent quality increase). The other variable that was manipulated was the mean time between arrivals of the tasks; we can then compare relative increases in quality based on the average utilization required by that set of tasks. When the average required utilization is greater than 1, it means that it is impossible to complete all tasks without approximating some of them.

There are three characterizations of this simulated environment that make it different from the actual environment of a system such as the DVMT:

- In interpretation environments like the DVMT, not all tasks need to be done, but in this simulation, in this section, they do. The rest of the chapter will discuss a more complex simulation where not all work needs to be done.
- This simulation assumes that approximating a result has only a local effect on quality, as opposed to reducing the quality of a whole group of related tasks.
- This simulation does not contain a model of the relationship between the fact that a task does or doesn't need to be done, and the fact that a facilitates relationship does or doesn't exist.

On the other hand, our measurements are against a design-to-time real-time scheduling algorithm that likewise does not take these factors into account. The addition of these extra characteristics to the task structure requires the addition of new coordination relationships (especially subtask). The second part of this chapter (starting at Section 6.2) will discuss GPGP using the full TÆMS task structure framework.

6.1.1 Calculating Delays

How do we compute how long to delay a task B that is facilitated by a task A? For example (see Figure 6.2), if

- utility is a function of result quality only (as opposed to a function of both quality and missed deadlines)
- the local scheduler does not assure the minimal quality of a task result
- *facilitates* affects only the durations of tasks directly

then agents must attempt to produce their highest quality solutions. Suppose there are several different tasks B_i such that A *facilitates* each B_i. If we estimate the latest start time for each task B in order to produce a highest quality solution, and estimate the finish time for task A, then all tasks B with latest start times after the estimated finish of A (B_2 and B_3 in the figure) should be delayed to the minimum latest start time of those tasks (B_2 in Figure 6.2; times must be calculated taking communication time into account if it is substantial). If the local scheduler *were* boundedly rational with respect to quality constraints, then the estimate of the latest start time of B could be modified by (made later by) the estimated effect of the *facilitates* relationship for the maximally assured minimal quality of the result of task A.

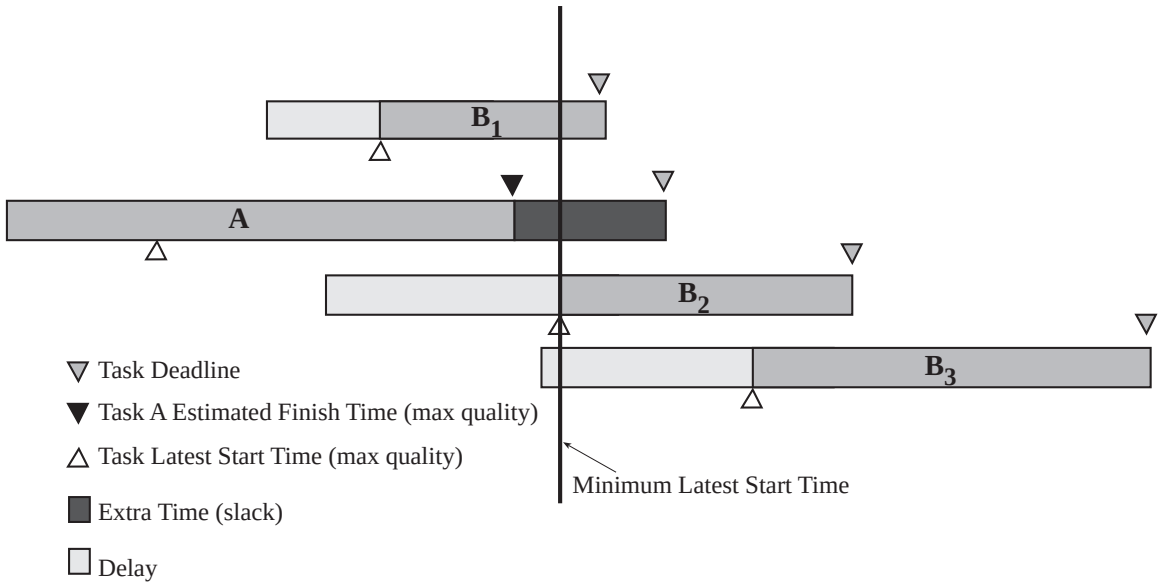

Figure 6.2. Calculating Delays

6.1.2 A Simple Model of the Utility of Detecting Facilitates

When should an agent test for a facilitation relationship? An agent can potentially test for the presence of the facilitates relationship between any new local task and the tasks it knows about from other agents. The costs per task here include the processing cost of testing for the relationship between two tasks and the cost of communication to get more information if the test result is too uncertain (C_{detect}). Note that the test itself is domain specific, even though the facilitates relationship is general. If the relationship is present, the benefits include

some increased utility (reduction in time, increase in quality of the facilitated task) (B_{fac}). The benefits will accrue fully only if the scheduling constraints implied by the new detected relationship can be incorporated, and only if the facilitating task is accomplished.[2] Costs are also incurred when the relationship is present, including the direct costs of communicating the result, adding the scheduling constraints, and updating the models of each others' tasks, and the indirect costs of delaying some tasks (C_{fac}). Using the *a priori* likelihood Pr_{fac} of a facilitates relationship from the environmental model, we can build an expression for the form of the expected utility of the relationship, $\text{Pr}_{\text{fac}} * (B_{\text{fac}} - C_{\text{fac}}) - C_{\text{detect}}$. In many environments, such as the DVMT, this expression will be relatively static, with C_{detect} relatively constant and with B_{fac} and C_{fac} depending on many environmental factors, including the task at hand, the current system utilization, etc. (see the experiments in Section 6.1.3). If C_{detect} is not relatively constant, then it is beneficial to consider separately the decision to detect a relationship and the decision to communicate about it.

6.1.3 When to Detect and Communicate Facilitates

The first suite of experiments (Figures 6.3 and 6.4) involves the effect of the two quantitative properties of facilitates, likelihood (Pr_{fac}) and duration power (ϕ_d). Our hypothesis is that the simple expected utility model given in Section 6.1.2 can be instantiated as a decision rule for each agent as to whether or not that agent should detect, communicate, and react to the facilitates relationship. The alternative is that the simple linear relationship is in fact not linear, or is drowned out by secondary characteristics such as the cluster size of the facilitates relationship (i.e. the length of a chain of facilitates relationships) or scheduler errors which cause the benefits of the facilitation relationship (B_{fac}) to become non-linear with respect to the power of the relationship.

To test this hypothesis, we first gathered raw data from 90 paired-response simulations of 4 agents that *never* detect, communicate, or react to the facilitates CR, and 4 agents that *always* detect, communicate, and react—5 simulations at each of 6 different duration powers and 3 different likelihoods (Figure 6.3). Each data point in the figure is the average of 20 computed percent quality increases (5 experiments of 4 paired agents each). All 90 experiments were simulated with the same frequency of task arrival (an average required utilization of 1.5—too high to allow all tasks to be completed at maximum quality without communication, but not so high as to saturate the DTT scheduler). Direct communication costs and detection costs were fixed at 0. Examining Figure 6.3, we find that below a power of 10%, in this example, exploiting the facilitates relationship costs more in *indirect* costs than it is worth. The indirect costs arise primarily from agent's delaying tasks and rearranging their schedules unnecessarily.

In Section 6.1.2 we postulated the form of the expected utility of detecting, communicating, and reacting to the CR as $\text{Pr}_{\text{fac}} * (B_{\text{fac}} - C_{\text{fac}}) - C_{\text{detect}}$. To instantiate this model for this experiment, we let $C_{\text{detect}} = 0$, make the benefits proportional to the duration power ($B_{\text{fac}} = \beta\phi_d$) and the costs constant (since the arrival rate was held constant) (C_{fac} = c). Applying linear regression we achieve a fit that explains 98% of the observed variance R^2 ($\beta = 1.013$, $c = 17.03$, both parameters are significant at the $\alpha = 0.01$ level). Thus this

[2] For some relationships, there is also a chance the benefit will accrue serendipitously without the detection of the relationship. When possible, tasks can be structured to take advantage of this fact, which may reduce coordination costs considerably.

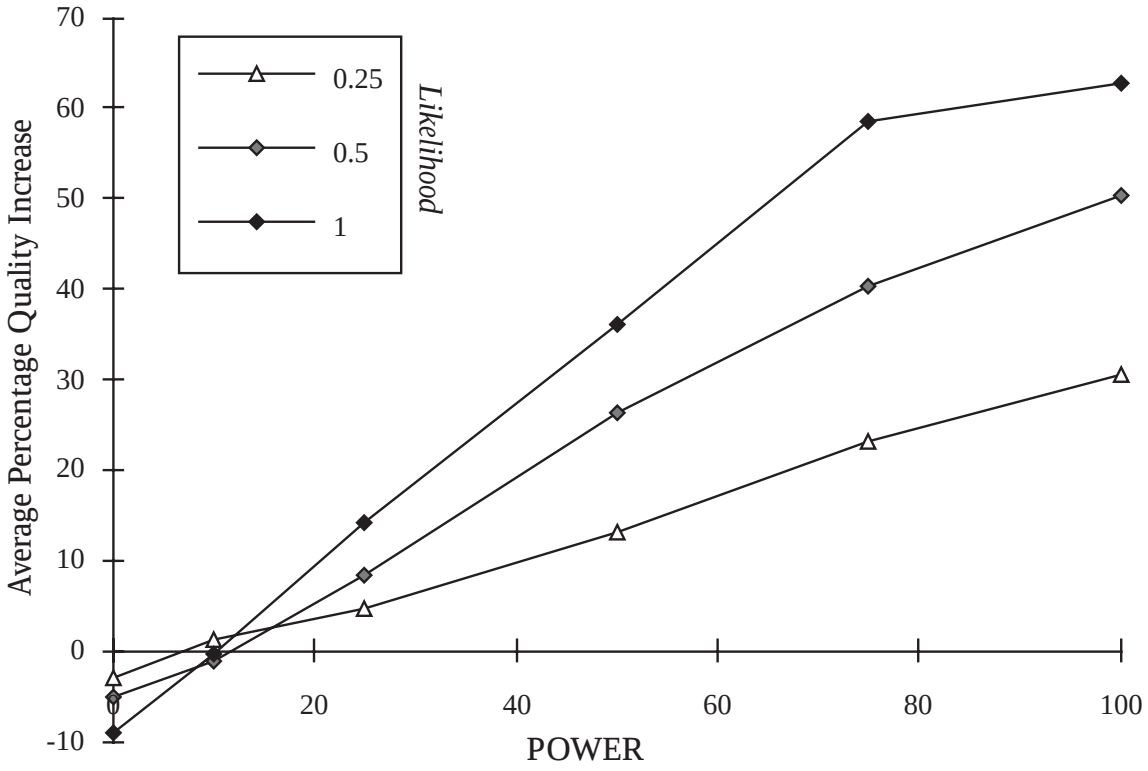

Figure 6.3. The effect of the power of the facilitates relationship on relative quality at different likelihoods

formula could be used as a decision rule by each agent to decide if it is worthwhile to detect, communicate, and react to the coordination relationship. This is another example of how different mechanisms are appropriate for different environments.

Figure 6.4 is a depiction of the surface defined by the three curves in Figure 6.3, via cubic interpolation. It succinctly shows how the positive effect of duration power grows with the likelihood. Note the flattening at the high end of the power and likelihood scales, caused by a ceiling effect since in this region all tasks are being accomplished at maximum quality (and so one cannot do any better).

6.1.4 Facilitating Real-time Performance

The second suite of experiments (Figure 6.5) shows that the average relative increase in quality in the results of communicating agents versus non-communicating agents grows with the required utilization of the system (load). The harder the task set is, the more important detecting the facilitates CR is, even at low power. The left side of Figure 6.5 shows duration power versus the relative quality increase for several required utilizations (system loads)[3]. The right side of Figure 6.5 shows the effect of required utilization on relative quality for duration

[3] Two data points on the utilization = 6 line, (power = 75, APQI = 555) and (power = 100, APQI = 1010), were left out for clarity.

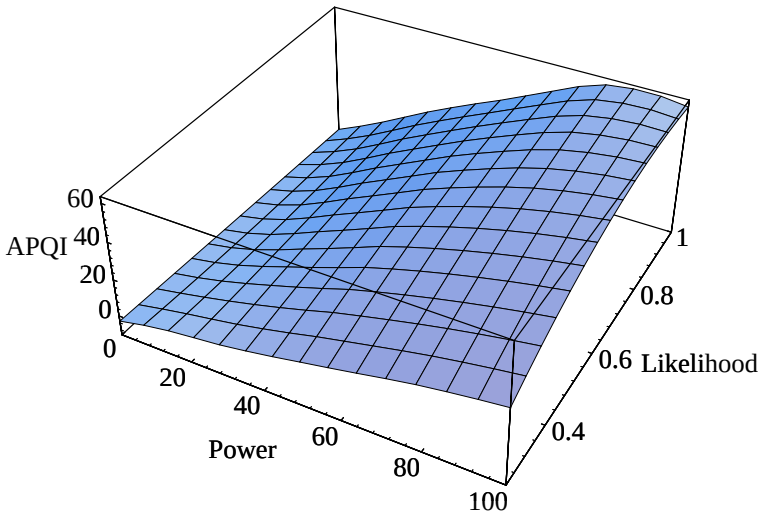

Figure 6.4. The effect of the power of the facilitates relationship on relative quality at different likelihoods

powers of 25% and 50%. The *a priori* likelihood of the presence of the CR was fixed at 0.5 for Figure 6.5. We would expect both the benefits and costs to change when the required utilization changes; the question is how much. The data from these experiments is not enough to disprove that the increase in quality grows linearly.

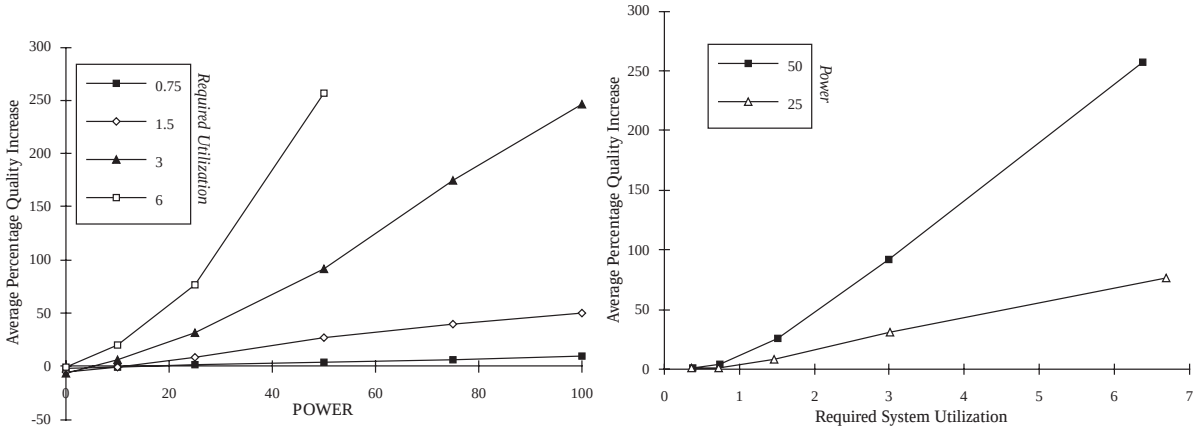

Figure 6.5. The effect of power and required utilization (system loads) on relative quality

These figures show the increase in quality, but just as important is the decrease in missed deadlines (not shown here), which is similar. At high loads (high required utilization) the

non-coordinated agent set reaches an asymptotic quality performance (which is less than maximum quality across loads; see Figure 6.6), and an unbounded number of missed deadlines. This figure depicts the absolute quality response of a single agent system over approximately 250 runs. The response has three major components: up to around a utilization of 1, absolute quality grows quickly as the number of tasks increases—each task is usually done at maximum quality; from around 1 to around 5, the DTT scheduler begins to trade off low quality, fast approximations for maximum quality, slow methods to avoid missing deadlines; above 5, every task is scheduled with the fastest method (resulting in asymptotic quality performance) and missed deadlines grow without bound.

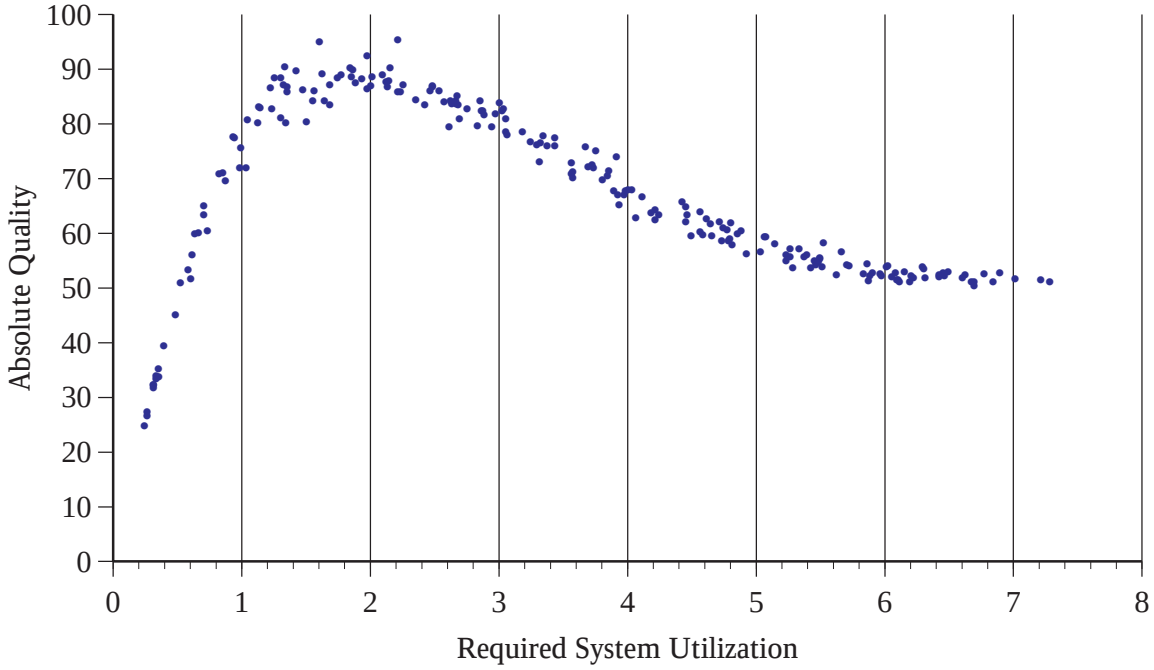

Figure 6.6. Effect of system load on the absolute quality for a one agent system

The effect of detecting, communicating about, and reacting to the facilitates CR is to move this curve upward. Even though the indirect costs of delaying tasks due to exploiting facilitation may increase under heavy loads, the average relative quality increase remains at 0 (rather than becoming negative) because the communicating agents can do no worse than the non-communicating agents, who are continuously executing tasks with the fastest, minimal quality method. The indirect costs can show up in more missed deadlines before the non-communicating schedulers become saturated near a utilization of 5. Figure 6.7 shows that while the *relative* performance of the coordinating agents grew in Figure 6.5, the *absolute* performance actually levels-off (note that the 'max-quality' line represents all tasks being completed at maximum quality, which is an impossible ideal to ever achieve for a required utilization greater than one).

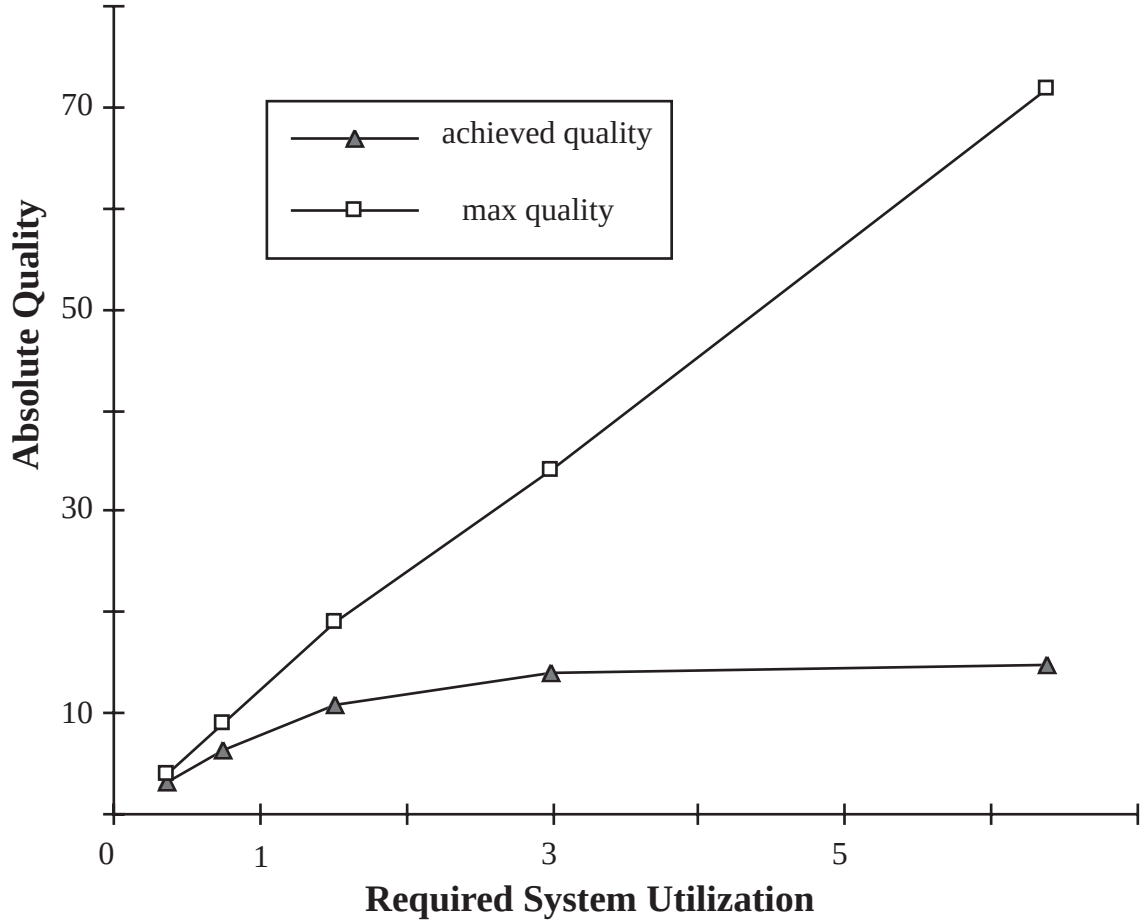

Figure 6.7. Effect of system load on the absolute quality

6.1.5 Delay

We ran a final suite of experiments to validate the effect of the delay time on performance. Assume again that task A facilitates tasks B_i as shown in Figure 6.8. The possible amounts by which to delay a task are bounded below (number 1 in Figure 6.8) by task A's estimated finish time for some quality, and bounded above (number 3 in Figure 6.8) by the minimum latest start time of all tasks B that can be started after A finishes *calculated to include the predicted effect of receiving the maximum quality result of task A*. For example, since the result of task A will reduce the amount of time required for tasks B_i, each task's latest start time would increase (move to the right) in Figure 6.2. Our choice, to delay to the minimum latest start time computed as if the result of A will *not* be received, is somewhere in between (number 2 in Figure 6.8).

We would like to show how this choice has the highest expected utility for the agent pair. Given an environment where power ϕ_d = 50%, likelihood $\mathrm{Pr}_{\mathrm{fac}}$ = 0.5, and a mean time between arrivals of 2.5, we achieve the following average percentage quality increases shown in Table 6.1.

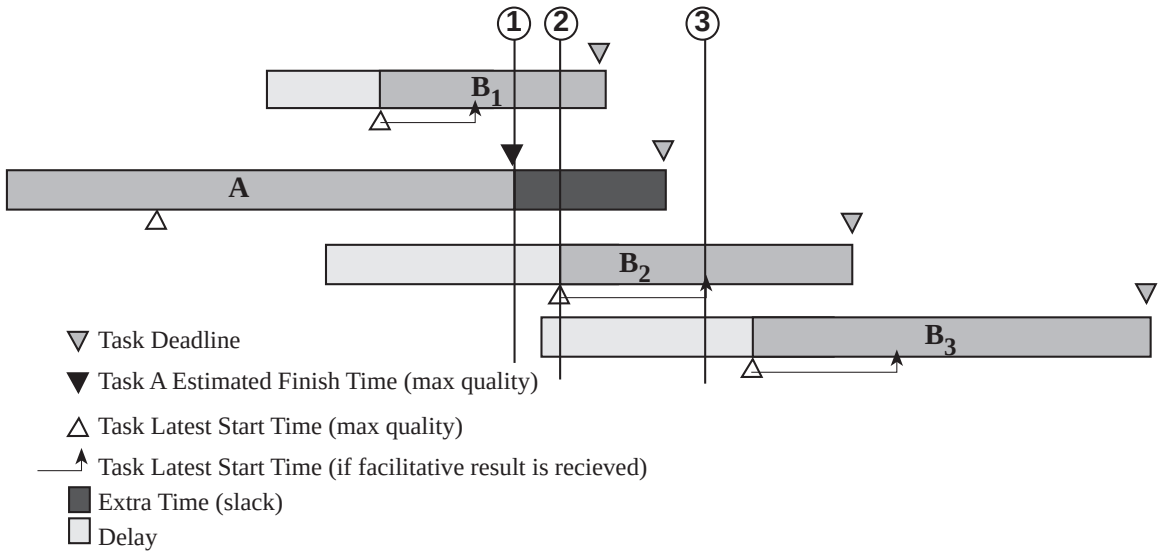

Figure 6.8. Other ways of calculating delays.

Table 6.1. Average percent quality increase for various commitment delay values.

	Shortest Delay (1)	Normal Delay (2)	Longest Delay (3)
APQI	19.1	26.4	22.9

When an agent commits to finishing task A early, it does so with a soft deadline—the task is scheduled to finish at the soft deadline time but monitoring will not switch to a faster, lower quality method unless the hard deadline is threatened. To commit to a hard deadline in order to take advantage of a facilitates relationship would be a complex decision, as the agent would have to weigh the cost of a potential local loss of quality with the potential gain in quality (through reduced duration) at the remote node. In general, shortening the delay hurts quality because often the *facilitating* task A does not quite finish on time (it does not have a hard deadline) and so the *facilitated* task begins without A's result. Lengthening the delay can also hurt quality because sometimes task A does not complete with the required quality (remember, the current scheduler is not boundedly-rational with respect to minimum needed quality), and so the quality of any delayed task B suffers. Delay time is similar to the slack time that was investigated in Durfee and Lesser's predictability vs. responsiveness experiments[Durfee and Lesser, 1988a].

This result gives some indication of one important design decision to be made in an agent's local scheduler—how long to delay execution of a facilitated method when there is no explicit commitment from the facilitating agent. GPGP mechanism 5, which coordinates soft predecessor relationships, solves this problem by communicating an explicit commitment. We will now move from these initial limited experiments to experiments with the full GPGP implementation.

6.2 GPGP Simulation: Issues

Our model of an abstract task environment, used in the rest of this chapter, has ten parameters; Table 6.2 lists them and the values used in the experiments described in the next two sections. These were discussed in some detail in Section 3.6.2. Figure 6.12 shows two small task groups generated with the parameters from Table 6.6, which are similar.

Table 6.2. Environmental Parameters used to generate the random episodes

Parameter	Values (facilitation exps.)	Values (clustering exps.)
Mean Branching factor (Poisson)	1	1
Mean Depth (Poisson)	3	3
Mean Duration (exponential)	10	(1 10 100)
Redundant Method QAF	Max	Max
Number of task groups	2	(1 5 10)
Task QAF distribution	(20%/80% min/max)	(50%/50% min/max) (100%/0% min/max)
Hard CR distribution	(10%/90% enables/none)	(0%/100% enables/none) (50%/50% enables/none)
Soft CR distribution	(80%/10%/10% facilitates/hinders/none)	(0%/10%/90% facilitates/hinders/none) (50%/10%/40% facilitates/hinders/none)
Chance of overlaps (binomial)	10%	(0% 50% 100%)
Facilitation Strength	.1 .5 .9	.5

The primary sources of overhead associated with the coordination mechanisms include action executions (communication and information gathering), calls to the local scheduler, and any algorithmic overhead associated with the mechanism itself. Table 6.3 summarizes the total amount of overhead from each source for each coordination mechanism setting and the coordination substrate. L represents the length of processing (time before termination), and d is a general density measure of coordination relationships. We believe that all of these amounts can be derived from the environmental parameters in Table 6.2, they can also be measured experimentally. Interactions between the presence of coordination mechanisms and these quantities include: the number of methods or tasks in $\mathbf{E}$, which depends on the non-local view mechanism; the number of coordination relationships $|\mathbf{CR}|$ or the subsets $\mathbf{RCR}$ (redundant coordination relationships), $\mathbf{HPCR}$ (hard predecessor coordination relationships), $\mathbf{SPCR}$ (soft predecessor coordination relationships), which depends on the number of tasks and methods as well; and the number of commitments $|\mathbf{C}|$, which depends on each of the three mechanisms that makes commitments.

6.3 General Performance Issues

Another question is how the performance of a fully configured system compares with optimal performance. While we have no optimal parallel scheduler with which to compare ourselves, we do have a single agent optimal scheduler and a centralized, heuristic parallel scheduler that takes the single-agent optimal schedule as its starting point.

We examined the general performance of the most complex (all mechanisms in place) and least complex (all mechanisms off) members of the GPGP family in comparison to each other,

Table 6.3. Overhead associated with individual mechanisms at each parameter setting

Mechanism *setting*	Communications	Information Gathering	Scheduler	*Other Overhead*
substrate	0	$\mathbf{E}$+*idle*	L	$O(L\mathbf{C})$
nlv *none*	0	0	0	0
some	$O(d\mathbf{P})$	$\mathbf{E}$*detect-CRs*	0	$O(T \in \mathbf{E})$
all	$O(\mathbf{P})$	$\mathbf{E}$*detect-CRs*	0	$O(T \in \mathbf{E})$
comm *min*	$O(\mathbf{C})$	0	0	$O(\mathbf{C})$
TG	$O(\mathbf{C} + \mathbf{E})$	0	0	$O(\mathbf{C} + \mathbf{E})$
all	$O(M \in \mathbf{E})$	0	0	$O(M \in \mathbf{E})$
redundant *on*	$O(\mathbf{RCR})$	0	0	$O(\mathbf{RCR} * S + \mathbf{CR})$
hard *on*	$O(\mathbf{HPCR})$	0	$O(\mathbf{HPCR})$	$O(\mathbf{HPCR} * S + \mathbf{CR})$
soft *on*	$O(\mathbf{SPCR})$	0	$O(\mathbf{SPCR})$	$O(\mathbf{SPCR} * S + \mathbf{CR})$

and in comparison to a centralized scheduler reference implementation (as an upper bound). We looked at performance measures such as the total final quality achieved by the system, the amount of work done, the number of deadlines missed, and the termination time. The centralized schedule reference system is not an appropriate solution to the general coordination problem, even for cooperative groups of agents, for several reasons:

- The centralized scheduling agent becomes a possible single point of failure that can cause the entire system to fail (unlike the decentralized GPGP system).
- The centralized scheduling agent requires a complete, global view of the episode—a view that we mentioned earlier is not always easy to achieve. We do not account for any costs in building such a global view in the reference implementation (viewing it as an upper bound on performance). We do not allow dynamic changes in the episodic task structure (which might require rescheduling).
- The centralized reference scheduler uses an *optimal* single-agent schedule as a starting point. The problem of scheduling actions in even fairly simple task structures is NP-complete, and the optimal scheduler's performance grows exponentially worse with the number of methods to be scheduled. Since the centralized reference scheduler has a global view and schedules all actions at all agents, the size of the centralized problem always grows faster than the size of the scheduling problems at GPGP agents with only partial views and heuristic schedulers.

We conducted 300 paired response experiments, using the three algorithms. "Balanced" refers to all mechanisms being on, with partial non-local views and communication of committed results and completed task groups. "Simple" refers to all mechanisms being off, with no non-local view and broadcast communication of all results. "Parallel" refers to the centralized reference scheduler that uses a heuristic parallelization of an optimal single agent schedule using a complete global view. The experiments were based on the same environmental parameters as the facilitation experiments (Table 6.2). There are several important things to note about this class of environments:

- The size of the episodes was kept artificially small so that the centralized reference scheduler could find an optimal schedule in a reasonable amount of run time.
- The experiments had very low (10%)numbers of enables relationships and a low (20%) number of MIN quality accrual functions because they penalize the simple algorithm—we demonstrate this in Section 6.8. High numbers of enables constraints tend to hurt the uncoordinated agents because they have no mechanism for handling precedence except 'wait until enabled'. If there is also a low overlap percentage, the uncoordinated agents, who do not exchange non-local views, will not be able to apply even this simple rule because they will not even know of the existence of the enabling method.
- Deadline pressure was also kept low (it also makes the simple algorithm perform badly).

In our experiments, the centralized parallel scheduler outperformed our distributed, GPGP agents 57% of the time (36% no difference, 7% distributed was better) using the total final quality as the only criterion. The GPGP agents produced 85% of the quality that the centralized parallel scheduler did, on average. These results need to be understood in the proper context—the centralized scheduler takes much more processing time that the distributed scheduler and can not be scaled up to larger numbers of methods or task groups. The centralized scheduler also starts with a global view of the entire episode. Table 6.4 shows the results for all four measured criteria by summarizing within-block (paired-response) comparisons. [4] For total final quality and number of deadlines missed, "better" simply refers to an episode where the algorithm in question had a greater total final quality or missed fewer deadlines, respectively. With respect to method execution time (a measure of system load) and termination time, "better" refers to the fact that one algorithm produced both a higher quality and missed fewer deadlines than the other algorithm, or if the two algorithms were the same, then the better algorithm had a lower total method execution time (lower load) or terminated sooner.[5]

We also looked at performance without any of the mechanisms; on the same 300 episodes the GPGP agents produced on average 1.14 times the final quality of the uncoordinated agents. Coordinated agents ("balanced") execute far fewer methods because of their ability to avoid redundancy. The redundant execution of methods proves a much more hindering element to the uncoordinated agents when acting under severe time pressure [Decker, 1994b]. Table 6.5 summarizes the results.

6.4 Taking Advantage of a Coordination Relationship: When to Add a New Mechanism

A practical question to ask is simply whether the addition of a particular mechanism will benefit performance for the system of agents. Here we give an example with respect to the soft coordination mechanism, which will make commitments to facilitation relationships. We ran 234 randomly generated episodes (generated with the environmental parameters shown in Table 6.2) with four agents both with and without the soft coordination mechanism. Because the variance between these randomly generated episodes is so great, we took advantage of the

[4] Communication can not be measured fairly because the centralized parallel scheduler had instantaneous communication to all agents and instantaneous access to all results.

[5] Termination within two time units was considered "the same" because the "balanced" algorithm has a fixed 2-unit startup cost. The average task duration is 10 time units.

Table 6.4. Performance comparison: Centralized Parallel Scheduler vs. Balanced GPGP Coordination and Decentralized DTT Scheduler

	Parallel better	Balanced Better	Same	*Significant?*
Total Final Quality	57%	7%	36%	yes
Method Execution Time	80%	7%	13%	yes
Deadlines Missed	1%	1%	98%	no
Termination Time	67%	15%	18%	yes

Table 6.5. Performance comparison: Simple GPGP Coordination vs. Balanced GPGP Coordination

	Simple better	Balanced Better	Same	*Significant?*
Total Final Quality	8%	21%	71%	yes
Method Execution Time	12%	72%	16%	yes
Deadlines Missed	0%	4%	96%	yes
Termination Time	9%	58%	33%	yes

paired response nature of the data to run a non-parametric Wilcoxon matched-pairs signed-ranks test [Daniel, 1978]. This test is easy to compute and makes very few assumptions—primarily that the variables are interval-valued and comparable within each block of paired responses. For each of the 234 blocks we calculated the difference in the total final quality achieved by each group of agents and excluded the blocks where there was no difference, leaving 102 blocks. We then replace the differences with the ranks of their absolute values, and then replace the signs on the ranks. Finally we sum the positive and negative ranks separately. A standardized Z score is then calculated. A small value of Z means that there was not much consistent variation, while a large value is unlikely to occur unless one treatment consistently outperformed the other. In our experiment, the null hypothesis is that the system with the soft coordination mechanism did the same as the one without it, and our alternative is that the system with the soft coordination mechanism did better (in terms of total final quality). The result here was $Z = -6.9$, which is highly significant, and allows us to reject the null hypothesis that the mechanism did not have an effect.

6.5 Different Family Members for Different Environments

In this section I will show a particular example of how different family members do better and worse in different environments. I will concentrate on two distinct family members—the 'modular agent' archetype (all CR modules on, non-local views, communicate commitments and completed task groups), and the 'simple agent' (no CR modules on, no non-local views, broadcast all completed methods). The environmental parameter we will vary (derived from the screening data collected in Section 6.8) is *QAF-min*, the percentage of tasks that have *min* as

their quality accumulation function ('AND' semantics). Our hypothesis was that the modular agents would do better than the simple agents as QAF-min increased (as more tasks needed to be done). We ran 250 paired-response experiments at 5 levels of QAF-min (0, 0.25, 0.5, 0.75, 1.0) with enables-probability varying also at the same 5 levels, no time pressure, overlaps of 0.5, 5 task groups, and 4 agents per run. The performance (in terms of total final quality) of the two coordination styles was significantly different by the Wilcoxon matched-pairs signed-ranks test (199 different pairs, $Z = -3.27$, $p \leq 0.0005$). More interestingly, we can see the difference in performance widening with the value of QAF-min. Figure 6.9 shows the probability of one coordination style or the other doing better (calculated simply from the frequencies) plotted verses the value of QAF-min. This allows you to see graphically the difference in the styles as QAF-min changes. Figure 6.10 shows the same information as probability partitions for each of the three possible outcomes as QAF-min changes, determined by (exploratory) logistic regression. The hash marks on the right edge of each box show the base partition values computed from all the data while ignoring the value of QAF-min.

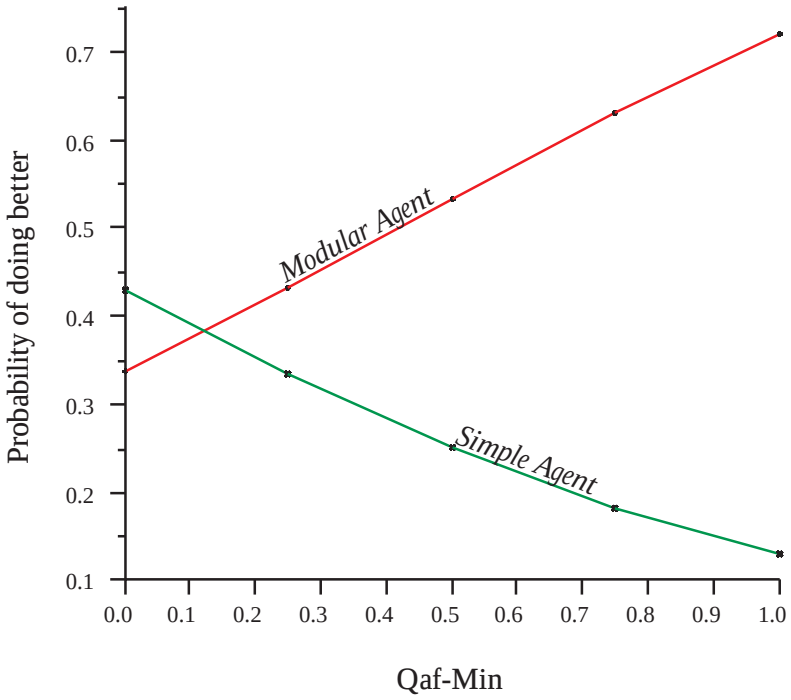

Figure 6.9. Plot of the probability of the modular or simple coordination styles doing better than the other (total final quality) verses the probability of task quality accumulation being MIN (AND-semantics)

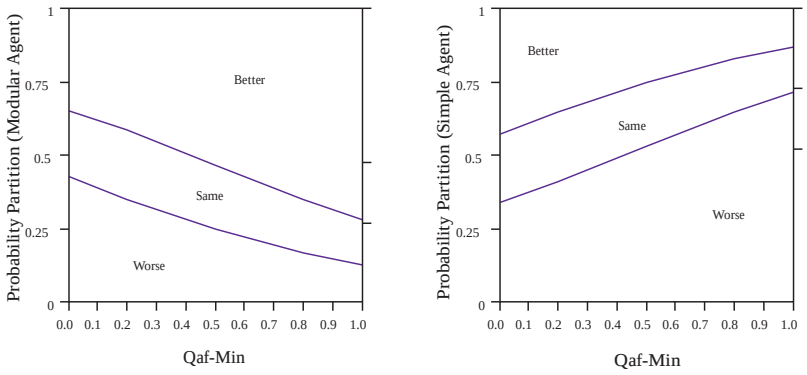

Figure 6.10. Probability partitions for one style doing the same, better, or worse than the other given the value of QAF-min.

6.6 Meta-level Communication: Return to Load Balancing through Dynamic Reorganization

Another question we have examined is the effect of task structure variance on the performance of load balancing algorithms. This work is a logical follow-on to the analysis of static, dynamic, and negotiated reorganization detailed in the last chapter. A *static* organization divides the load up *a priori*—in the case below, by randomly assigning redundant tasks to agents. A *one-shot dynamic* reorganization, like that analyzed in Chapter 4, assigns redundant tasks on the basis of the *expected* load on other agents. A *meta-level communication* (MLC) reorganization assigns redundant tasks on the basis of actual information about the particular problem-solving episode at hand. Because it requires extra communication, the MLC reorganization is more expensive, but the extra information pays off as the *variance* in static agent loads grows.[6]

A MLC coordination mechanism (mechanism 6) can be implemented in GPGP. Many such implementations are possible; the one that we chose works by altering the way redundant commitments are handled. When a commitment is sent to another agent, it is modified to include the current *load* of the agent making the commitment (to be precise, the amount of work for the agent in the current schedule). Whenever a decision about redundant commitments need to be made at another agent (in mechanisms 3, 4, and 5—simple redundancy, hard, and soft successor relationship handling) the load of the agents with the redundant commitments are taken into account at the point where ties would have been broken randomly. The agent with the lowest load keeps the commitment instead. If the loads are equal, the tie is broken randomly as before.

The effect of this mechanism on the general GPGP environments when agents use the default Design-To-Time scheduler is minimal. The heuristics used by the DTT scheduler are focused at providing the highest possible total final quality for the agent without violating deadlines—this is not the same as terminating quickly, and the scheduler has no heuristics to

[6] See also Mirchandaney's [Mirchandaney *et al.*, 1989] paper on the effects of delays on load balancing in simple distributed systems.

prefer earlier termination times (nor, frankly, should it have them). In a randomly-generated task environment, where the methods are assigned to agents randomly (and therefore, somewhat evenly) there is rarely any significant change in termination time.

However, if you recall one of our results from Chapter 4, you will remember that MLC coordination is most useful in environments with high *variance* in the task structures presented to agents. We can look at our experiments in this light, by calculating an endogenous input variable for each run that represents the amount of variance in redundant tasks (the ones that would *potentially* be eligible for a load-balancing mechanism decision). Figure 6.11 shows how the probability of terminating more quickly with the MLC load balancing algorithm grows as the standard deviation in the total durations of redundant tasks at each agent grows.

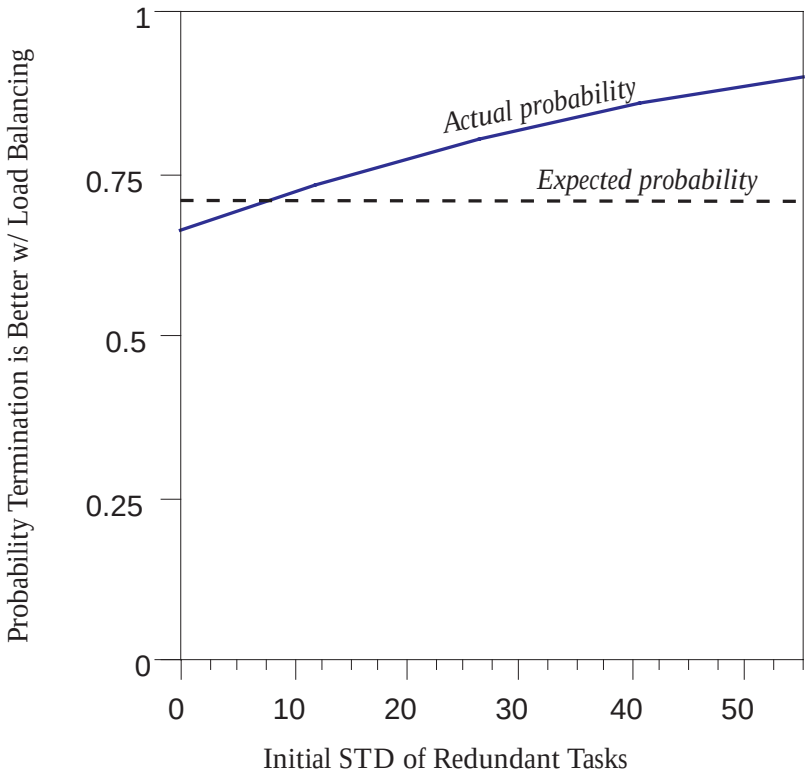

Figure 6.11. Probability that MLC load balancing will terminate more quickly than static load balancing, fitted using a loglinear model from actual TÆMS simulation data.

6.7 Computational Organizational Design

Another application for the use of TÆMS as a modeling tool for asking and answering questions about organizational design, as was discussed in Chapter 3. In this final section we replicate a classic computational organization design study done by Burton and Obel ([Burton and Obel, 1984]) using GPGP agents.

6.7.1 Burton and Obel Experiments

In this section we will consider the use of TÆMS as a simulator to explore hypotheses about the interactions between environmental and agent-structural characteristics. We use as an example a question explored by Burton and Obel: is there a significant difference in performance due to either the choice of organizational structure or the decomposability of technology[7]?

We equate a technology with a TÆMS task structure, instead of a linear program. Task structures allow us to use a clear interval measure for decomposability, namely the probability of a task interrelationship (in this example enables, facilitates, and overlaps). We define a nearly decomposable task structure to have a base probability of 0.2 for these three coordination relationships and a less decomposable task structure to have a base probability of 0.8 (see Figure 6.12). We will continue in this example to look at purely computational task structures, although the use of physical resources can also be represented (see [Decker and Lesser, 1993e]).

Burton and Obel were exploring the difference in M-form (multidivisional) and U-form (unitary—functional) hierarchical structures; we will analyze the GPGP family of team-oriented coordination algorithms. For our structural variable we will vary the communication of non-local views (GPGP Module 1). Informally, we will be contrasting the situation where each agent makes commitments and communicates results based only on local information (*no* non-local view) with one where the agents freely share task structure information with one another across coordination relationships (*partial* non-local view). Figure 6.13 shows an example—note that in neither case does the agent have the *global* view of Figure 6.12.

Burton and Obel used a profitability index as their performance measure, derived from the percentage of optimal profit achieved. In general, the scheduling an arbitrary TÆMS task structure is an NP-hard problem and so we do not have access to optimal solutions. Instead we compare performance directly on four scales: the number of communication actions, the amount of time spent executing methods, the final quality achieved, and the termination time. Simulation runs for each of the four combinations of non-local view policy and level of task decomposability were done in matched sets—the randomly generated episode was the same for each combination with the exception of more coordination relationships (including more overlapping methods) being added in the less decomposable task structures. Following Burton and Obel, we used the non-parametric Friedman two-way analysis of variance by ranks test for our hypotheses. The assumptions of this test are that each block (in our case, randomly generated episode) is independent of the others and that each block contains matched observations that may be ranked with respect to one another. The null hypothesis is that the populations within each block are identical.

We generated 40 random episodes of a single task group, each episode was replicated for the four combinations in each block. We used teams consisting of 5 agents; the other parameters used in generating the task structures are summarized in Table 6.6 and a typical randomly generated structure is shown in Figure 6.12. Figure 6.13 shows the difference in the local view of one agent with and without creating partial non-local views. We first tested two major hypotheses:

---

[7] *Technology* is used here in the management science sense of "the physical method by which resources are converted into products or services" or a "means for doing work" [Burton and Obel, 1984, Scott, 1987].

Table 6.6. Parameters used to generate the 40 random episodes

Parameter	Value
Mean Branching factor (Poisson)	1
Mean Depth (Poisson)	3
Mean Duration (exponential)	10
Redundant Method QAF	Max
Number of task groups	1
Task QAF distribution	(50% min) (50% max)
Decomposition parameter	p = 0.2 or 0.8
Hard CR distribution	(p enables) ((1-p) none)
Soft CR distribution	(p facilitates) (10% hinders) ((.9-p) none)
Chance of overlaps (binomial)	p

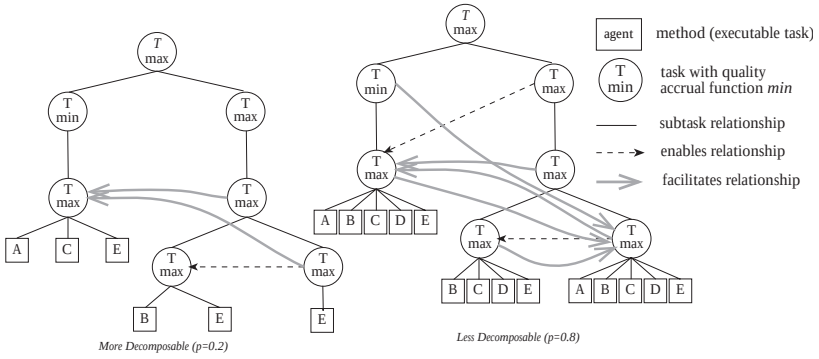

Figure 6.12. Example of a randomly generated objective task structure, generated with the parameters in the previous table.

Hypothesis 1: *There is no difference in performance between agents with a partial non-local view and those without.* For the communication and method execution performance measures, we reject the null hypothesis at the 0.001 level. We cannot reject the null hypothesis that there is no difference in final quality and termination time. Teams of computational agents that exchange information about their private, local views consistently exchange more messages (in this experiment, a mean increase of 7 messages) but do less work (here, a mean decrease of 20 time units of work, probably due mostly to avoiding redundancy).

Hypothesis 2: *There is no difference in performance due to the level of decomposability of technology.* For the communication and method execution performance measures, we reject the null hypothesis at the 0.001 level. We cannot reject the null hypothesis that there is no difference in final quality and termination time. Teams of computational agents, regardless of their policy on the exchange of private, local information communicate more messages (in this experiment, a mean increase of 47 messages) and do more work (here, a mean increase of 24 time units) when faced with less decomposable computational task structures (technology).

Again following Burton and Obel, we next test for interaction effects between non-local view policy and level of technology decomposability by calculating the differences in performance at each level of decomposability, and then testing across non-local view policy. This test was not significant. To reiterate, based on the experimental results in this section it seems that TÆMS has potential for use in computational organization theory, though this is out of the scope of this dissertation.

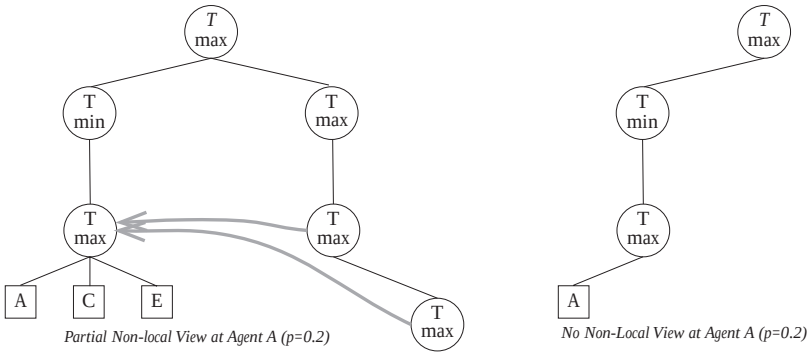

Figure 6.13. Example of the local view at Agent A when the team shares private information to create a partial non-local view and when it does not.

6.8 Exploring the Family Performance Space

Finally, we looked at the multidimensional performance space for the family of coordination algorithms over four different performance measures. At the most abstract level, each of the five mechanisms are parameterized independently (the first two have three possible settings and the last three can be 'in' or 'out') for a total of 72 possible coordination algorithms.

Clustering can be a useful method for dealing with large algorithm spaces to prune search for an appropriate combination of mechanisms. We applied two standard statistical clustering techniques to develop a much smaller set of significantly different algorithms. The resulting five 'prototypical' combined behaviors are a useful starting point when searching for an appropriate algorithm family member in a new environment.

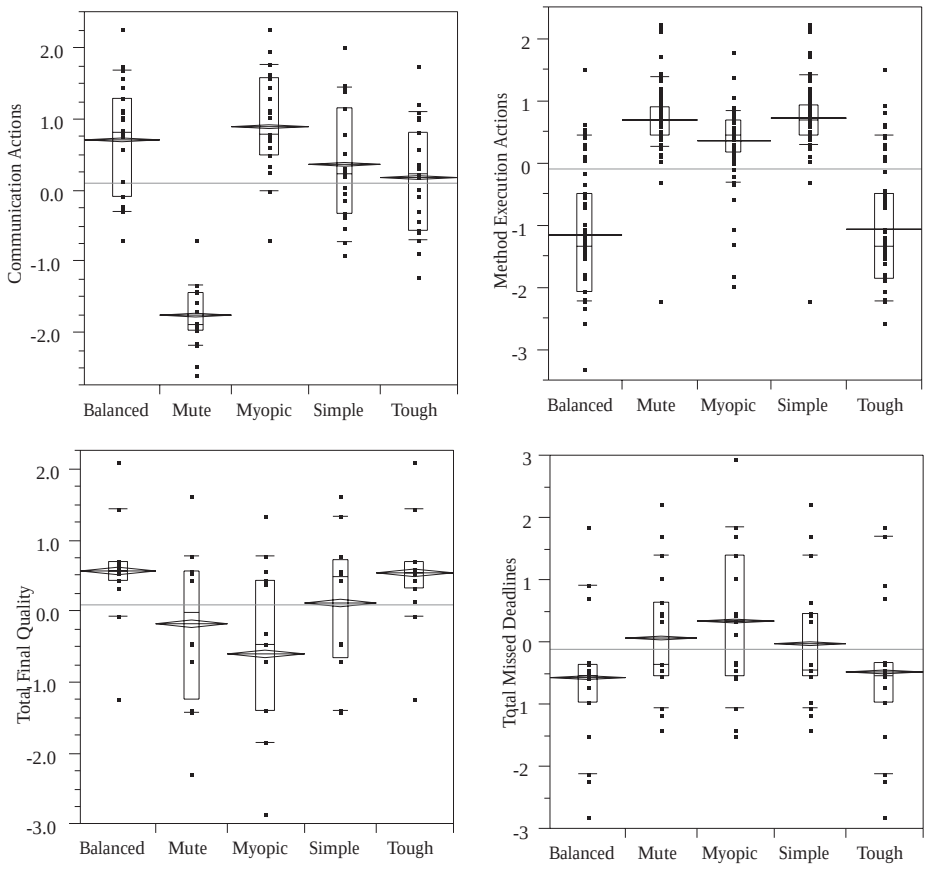

Figure 6.14. Standardized Performance by the 5 named coordination styles.

The analysis proceeded as follows: we generated one random episode in each of 63 randomly chosen environments, and ran each of the 72 "agent types" on the episode (4536 cases). We collected four performance measures: total quality, number of methods executed, number of communication actions, and termination time. We then took this data and standardized each performance measure within an environment. So now each measure is represented as the number of standard deviations from the mean value in that environment. We then took summary statistics for each measure grouped by agent types—this boils the 4536 cases (standardized within each environment) into 72 summary cases (summarized across environments). Each of the 72 summaries correspond to the average standardized performance of one agent-type for the four performance measures. We then used both a hierarchical

clustering algorithm (SYSTAT JOIN with 'complete linkage'[8]) to produce the following general prototypical agent classes (we chose one representative algorithm in each class):

Simple: No commitments or non-local view, just broadcasts results.

Myopic: All commitment mechanisms on, but no non-local view.

Balanced: All mechanisms on.

Tough-guy: Agent that makes no soft commitments.

Mute: No communication whatsoever[9]

Figure 6.14 shows the values of several typical performance measures for only the five named types. Performance measures were standardized *within each episode*, (i.eacross all 72 types). Shown for each are the means and 10, 25, 50, 75, and 90 percent quantiles. All agents' performances are significantly different by Tukey Kramer HSD except for: Method Execution (Simple vs. Mute), Total final quality (Balanced vs. Tough), Deadlines missed (simple vs. mute) and (balanced vs. tough).

We are also analyzing the effect of environmental characteristics on agent performance. Figure 6.15 shows an example of the effect of the *a priori* amount of overlap on the number of method execution actions for the five named agent types. Note again that the balanced and tough agents do significantly less work when there is a lot of overlap (as would be expected). The performance of the tough and balanced agents is similar because (from Table 6.2) only half the experiments had any facilitation, and when it was present was only at 50% power.

A linear clustering algorithm, SYSTAT KMEANS, produces a similar result as hierarchical clustering, and also produces the mean value of each performance measure for each group, so you can in fact see that the non-communicating agents have a high negative mean "number-of-communications" (-1.16–remember these were averaged from standardized scores) but execute more methods on average and produce less final quality. They also terminate slightly quicker than average. Our "Balanced" group, in comparison, communicates a little more than average, executes *many* fewer methods (-1.29—way out on the edge of this statistic), returns better-than-average quality and about average termination time. This is reasonable, as 'avoiding redundant work' and other work-reducing ideas are a key feature of the GPGP algorithm. For example purposes, complete KMEANS output for 48 of the agent types[10] is listed in Tables 6.7 and 6.8. Each of the 48 agent types is summarized by a alphabetic genotype. The first letter is 'H' if mechanism 5 (Hard relationships) is on and '–' if it is off. The second letter is 'S' if mechanism 4 (Soft relationships) is on and '–' if it is off. The third letter is 'R' if mechanism 3 (simple Redundancy) is on and '–' if it is off. The fourth letter indicates the state of mechanism 1 (non-local views): 's' for some and 'n' for none. The fifth and final letter indicates the state of mechanism 2 (result communication): 'A' for all, 'C' for commitments only, and 'T' for commitments + task groups. Note that the KMEANS clustering groups 'simple' and 'myopic' together, and puts '– – – s T' and '– – – s C' in a different cluster. I chose to keep 'simple'

[8] Distances are calculated between the farthest points in each cluster. Other distance measures (Euclidean, centroid, or Pearson correlation) gave similar results.

[9] This algorithm makes no commitments (mechanisms 3, 4, and 5 off) and communicates (mechanism 2) only 'satisfied commitments'—therefore it sends no communications ever!.

[10] Agents with mechanism 1 (non-local views) set to 'all' were accidentally omitted.

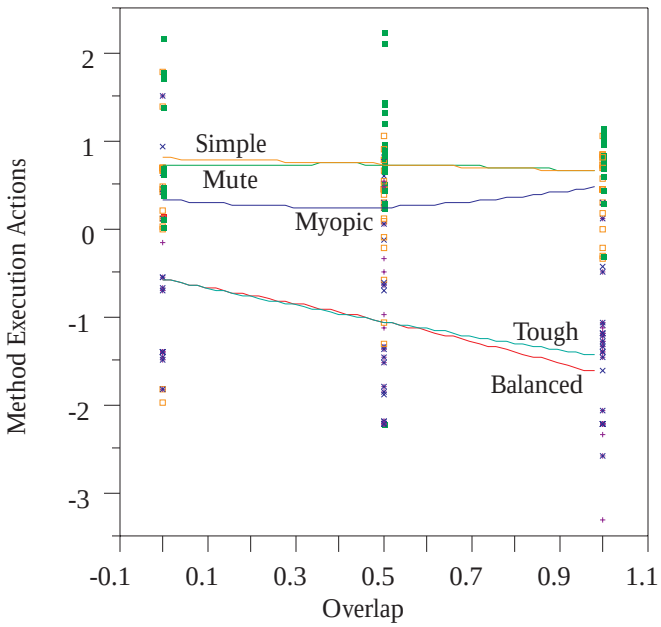

Figure 6.15. The effect of overlaps in the task environment on the standardized method execution performance by the 5 named coordination styles (smoothed splines fit to the means).

and 'myopic' separated because they are different in an interesting way ('myopic' has every mechanism on except for the exchange of non-local views), whereas the two others are basically as uninteresting as 'Mute' except that they exchange non-local views which fool them into doing no work and then waiting around forever (until the deadline) for the results to be sent (which or course never happens since only committed results are communicated and no commitments are made).

6.9 Summary

In this chapter I described several experiments using TÆMS and the GPGP algorithm. This chapter differs from Chapter 4 in that the focus here is on examining the performance of computer programs, rather than the abstract mathematical analysis of a problem that uses computer programs only for verification. First this chapter described some simple experiments showing the effect of facilitation power and likelihood on agents; the rest of the chapter focused on Generalized Partial Global Planning.

The chapter showed how to decide if the addition of a new GPGP mechanism was useful. It showed the general performance of two GPGP family algorithms compared to a centralized parallel reference algorithm; GPGP with all mechanisms 'on' produces 85% of the quality of the centralized reference scheduler implementation in a random environment. Such performance is reasonable and we feel could be made better by developing better local scheduling algorithms and new coordination mechanisms.

This chapter contained a demonstration of how a feature of the task environment (the probability of task quality accumulation being MAX) can cause different GPGP family members to be preferred. This chapter also discussed a sixth mechanism, a load balancing mechanism

Table 6.7. Complete KMEANS linear clustering output for all 72 agent types, first three clusters. All performance parameters were standardized within blocks.

CLUSTER NUMBER: 1						
Members		Statistics				
CASE	DISTANCE	VARIABLE	MINIMUM	MEAN	MAXIMUM	ST.DEV.
SIMPLE - - - nA	0.30	Comm	0.06	0.55	1.20	0.30
- - RnA	0.17	MEA	-0.34	0.29	0.57	0.28
-S-nA	0.17	TFQ	-0.24	0.09	0.42	0.23
-SRnA	0.25	Term	-0.35	-0.02	0.14	0.14
H- - nA	0.16					
H-RnA	0.22					
HS-nA	0.22					
HSRnA	0.36					
- - - sA	0.16					
-S-sA	0.30					
H- - sA	0.26					
HS-sA	0.36					
- - RnT	0.32					
-SRnT	0.14					
H-RnT	0.13					
Myopic HSRnT	0.20					
HSRnC	0.24					
CLUSTER NUMBER: 2						
Members		Statistics				
CASE	DISTANCE	VARIABLE	MINIMUM	MEAN	MAXIMUM	ST.DEV.
- - - sT	0.32	Comm	-1.26	-1.03	-0.80	0.23
- - - sC	0.32	MEA	-0.02	-0.02	-0.02	0.00
		TFQ	-0.37	-0.37	-0.37	0.00
		Term	1.27	1.88	2.48	0.60
CLUSTER NUMBER: 3						
Members		Statistics				
CASE	DISTANCE	VARIABLE	MINIMUM	MEAN	MAXIMUM	ST.DEV.
-S-nT	0.26	Comm	-0.94	-0.49	-0.07	0.28
H- - nT	0.15	MEA	-0.34	-0.01	0.56	0.31
HS-nT	0.35	TFQ	-0.53	0.03	0.63	0.42
-S-sT	0.21	Term	-0.29	-0.05	0.15	0.16
Tough H- - sT	0.27					
HS-sT	0.41					
-SRnC	0.32					
H-RnC	0.27					
HS-nC	0.34					
-S-sC	0.30					
H- - sC	0.35					
HS-sC	0.36					

Table 6.8. Complete KMEANS linear clustering output for all 72 agent types, continued. All performance parameters were standardized within blocks.

CLUSTER NUMBER: 4						
Members		Statistics				
CASE	DISTANCE	VARIABLE	MINIMUM	MEAN	MAXIMUM	ST.DEV.
- - RsA	0.21	Comm	-0.37	0.45	1.27	0.48
-SRsA	0.27	MEA	-1.49	-1.29	-1.12	0.14
H-RsA	0.36	TFQ	-0.35	0.25	0.63	0.40
HSRsA	0.47	Term	-0.22	-0.01	0.20	0.16
- - RsT	0.40					
-SRsT	0.21					
H-RsT	0.21					
Balanced HSRsT	0.27					
- - RsC	0.52					
-SRsC	0.31					
H-RsC	0.31					
HSRsC	0.22					
CLUSTER NUMBER: 5						
Members		Statistics				
CASE	DISTANCE	VARIABLE	MINIMUM	MEAN	MAXIMUM	ST.DEV.
- - - nT	0.18	Comm	-1.68	-1.16	-0.75	0.33
Mute - - - nC	0.27	MEA	0.15	0.43	0.57	0.17
- - RnC	0.24	TFQ	-0.31	-0.25	-0.20	0.03
-S-nC	0.07	Term	-0.35	-0.19	0.03	0.13
H- - nC	0.16					

that communicates meta-level information, and showed that it was somewhat more useful when the variance in duration of the agents' overlapping tasks was high. This section thus ties-in back to the discussion at the end of Chapter 4 on the usefulness of meta-level communication (in this case, the transmission of local load information) when the inter-episode variance (in this case, in the initial agent loads) is high.

The chapter also presented a version of Burton and Obel's task decomposability experiment that used GPGP agent teams instead of hierarchical organizational structures. As in Burton and Obel's result, we showed that both the task structure decomposability and the organizational structure (in our case, the state of the non-local view communication mechanism) have a significant effect.

Finally, we gave a sense of the performance space of the 5 broadly-parameterized mechanisms using a clustering technique. Clustering can be a useful method for dealing with large algorithm spaces to prune search for an appropriate combination of mechanisms. Such methods may also lead to ways to learn situation-specific knowledge about the application of certain mechanisms in certain situations (perhaps using case-based reasoning techniques).

Some of the things that are in this chapter that I talked about in Chapter 1 include doing paired-response comparisons of different coordination algorithms in Section 6.2, doing ablation studies of the addition/deletion of mechanisms and its effect on performance (Section 6.8) and examining computerized methods for organizational design (Section 6.7). Finally, this chapter described several sets of experiments that dealt with the performance of the GPGP mechanisms in different environments, using a real implementation (running in Common Lisp on multiple platforms, and being used by more than one person). The implementation uses as submodules a real implementation of Garvey's Design-To-Time real-time scheduler [Garvey and Lesser, 1994].

We believe that GPGP can become a reusable, domain-independent basis for multi-agent coordination when used in conjunction with a library of coordination mechanisms and a learning mechanism. We intend to develop such a library of reusable coordination mechanisms. For example, mechanisms that work from the successors of hard and soft relationships instead of the predecessors, negotiation mechanisms, mechanisms for behavior such as contracting, or mechanisms that can be used by self-motivated agents in non-cooperative environments. Many of these mechanisms can be built on the existing work of other DAI researchers. Future work will also examine expanding the parameterization of the mechanisms and using machine learning techniques to choose the appropriate parameter values (i.e., learning the best mechanism set for an environment). Finally, we are also beginning work on using the GPGP approach in applications ranging from providing human coordination assistance to distributed information gathering. This will be discussed in the next chapter in the Future Work section. The next chapter, which is the final chapter, will summarize the previous chapters and discuss the extremely interesting new areas of research to which this dissertation leads us.

CHAPTER 7

CONCLUSIONS

> *From decentralization we get responsibility, development of personnel, decisions close to the facts, flexibility—in short, all of the qualities necessary for an organization to adapt to new conditions. From coordination we get efficiencies and economies. It must be apparent that coordinated decentralization is not an easy concept to apply.*
>
> *— Alfred P. Sloan, Jr., My Years with General Motors*

Coordination is the process of managing interdependencies between activities [Malone and Crowston, 1991]. This dissertation focused on the problem of representing these interdependencies in a formal, domain-independent way.

This dissertation demonstrated a framework that can be used to specify the task structure of any computational environment, called TÆMS. It then instantiated an existing methodology (MAD: Modeling, Analysis and Design [Cohen, 1991]) using this framework to analyze a particular computational environment (Distributed Sensor Networks) and predict and verify the performance of two simple coordination algorithms in that environment. Finally, we designed a family of generic coordination mechanisms for cooperative, soft real-time computational task environments and demonstrated their performance and why a *family* of mechanisms is needed instead of a single static algorithm.

7.1 Summary: Representing Task Environments

This dissertation was about how to represent coordination problems in a formal, domain-independent way. Such a representation should abstract out the details of the domain, and leave the basic coordination problem—the choice and temporal ordering of possible actions. Our solution to this problem was a framework called TÆMS (Task Analysis, Environment Modeling, and Simulation), which can be used to specify, reason about, analyze, and simulate any computational environment. The unique features of TÆMS include:

- The explicit, quantitative representation of task interrelationships. Both hard and soft, positive and negative relationships can be represented. When relationships in the environment extend between tasks being worked on by separate agents, we often call them *coordination relationships*, because they can be used to design and analyze coordination mechanisms. We demonstrated the usefulness of coordination relationships in creating coordination mechanisms in Chapters 5 and 6. We gave definitions for fifteen useful relationships in Chapter 3. The set of relationships is extensible.
- The representation of structures at multiple levels of abstraction. The lowest level of abstraction is called an executable *method*. A *method* 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 can 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.

- The representation of structure from three different viewpoints. The first view is a *generative* model of the problem solving episodes in an environment—a statistical view of the task structures. The second view is an *objective* view of the actual, real, instantiated task structures that are present in an episode. The third view is the *subjective* view that the agents have of the objective task structures.

- TÆMS makes very few assumptions about what an 'agent' is. TÆMS defines an agent as a locus of subjective belief (or state) and action (executing methods, communicating, and acquiring subjective information about the current problem solving episode). This is important because the study of principled agent construction is a very active area. By separating the notion of agency from the model of task environments, we do not have to subscribe to particular agent architectures (which one would assume will be adapted to the task environment at hand), and we may ask questions about the inherent social nature of the task environment at hand (allowing that the concept of society may arise before the concept of individual agents [Gasser, 1991]). Such a conception is unique among computational approaches.

- TÆMS allows us to clearly specify concepts and subproblems important to multi-agent and AI scheduling approaches. For example, we discussed the difference between "anytime" and "design-to-time" algorithms using TÆMSin Chapter 3. Garvey [Garvey *et al.*, 1993] uses TÆMS to define the concept of a minimum duration schedule that achieves maximum quality.

- This dissertation is about *computational* task environments, where methods are things like blackboard knowledge source instantiations. However, we also described extensions of TÆMS to represent physical resource constraints.

Along with its formal definition, we also described a simulator for TÆMS written in portable CLOS (the Common Lisp Object System) and using CLIP[Westbrook *et al.*, 1994] for data collection.

We validated this framework by building a detailed model of the complex DSN environment of the Distributed Vehicle Monitoring Testbed (DVMT). Our model included features that represent approximate processing, faulty sensors and other noise sources, low quality solution errors, sensor configuration artifacts, and vehicle tracking phenomena such as training and ghost tracks. Simulations of simplified DSN models showed many of the same characteristics as were seen in the DVMT [Durfee *et al.*, 1987]. We also described models of many other environments: hospital patient scheduling, a post office problem, airport resource management, multi-agent Internet information gathering, and pilot's associate. Finally, we validated our framework by allowing others to use it in their work—on design-to-time scheduling, on parallel scheduling, on the diagnosis of errors in local area networks, and in the future to model software engineering activities.

7.2 Summary: Analyzing a Distributed Sensor Network Environment

The second major result reported here was a detailed analysis of a simplified DSN environment. The methodology behind this analysis is an instantiation of the MAD (Modeling, Analysis, and Design) methodology [Cohen, 1991], with TÆMS providing the modeling and simulation components. This part of the dissertation returns to the work of Durfee, Lesser, and Corkill [Durfee *et al.*, 1987] that showed that no single coordination algorithm uniformly outperformed the others. This dissertation explained that result, and went on to predict the performance effects of changing:

- the number of agents
- the physical organization of agents (i.e., the range of their sensors and how much the sensed regions overlap)
- the average number of vehicles in an episode
- the agents' coordination algorithm
- the relative cost of communication and computation

These predictions were verified by simulation.

For example, in Chapter 4 we derived and verified an expression for the time of termination of a set of agents in any arbitrary simple DSN environment that has a static physical organization and coordination algorithm. We used this expression (Eq. 4.9) as a predictor by combining it with the probabilities for the values of $\hat{S}$ and $\hat{N}$. We verified this model using the simulation component of TÆMS.

Our analysis explained another observation that has been made about the DVMT—that the extra overhead of meta-level communication is not always balanced by better performance. This work represents the first detailed analysis of a DSN, and the first quantitative, statistical analysis of any Distributed AI system outside Sen's work on distributed meeting scheduling for two agents [Sen and Durfee, 1994].

7.3 Summary: Designing a Family of Coordination Mechanisms

The third major result reported here was the design and evaluation of a family of coordination mechanisms for cooperative computational task environments. We called the collection of resulting algorithms the Generalized Partial Global Planning (GPGP) family of algorithms. GPGP both generalizes and extends Durfee's Partial Global Planning (PGP) algorithm [Durfee and Lesser, 1991]. Our approach has several unique features:

- Each mechanism is defined as a response to certain features in the current subjective task environment. Each mechanism can be removed entirely, or parameterized so that it is only active for some portion of an episode. New mechanisms can be easily defined. An initial set of five mechanisms was examined and we added a sixth (a simple load balancing mechanism).
- GPGP works in conjunction with an existing agent architecture and local scheduler. The experimental results reported here were achieved using a 'design-to-time' soft real-time local scheduler developed by Garvey [Garvey and Lesser, 1993].

- GPGP, unlike PGP, is not tied to a single domain.
- GPGP allows more agent heterogeneity than PGP with respect to agent capabilities.
- GPGP mechanisms in general exchange less information than the PGP algorithm, and the information that GPGP mechanisms exchange can be at different levels of abstraction. PGP agents generally communicate complete schedules at a single, fixed level of abstraction. GPGP mechanisms communicate scheduling commitments to particular tasks, at any convenient level of abstraction.

The fact that most of the GPGP coordination mechanisms use commitments to other agents as local scheduling constraints is the reason that the GPGP family of algorithms requires cooperative agents. Nothing in TÆMS the underlying task structure representation, requires agents to be cooperative, antagonistic, or simply self-motivated. We could develop or co-opt mechanisms to enforce the keeping of agent commitments (for instance, as contracts [Sandholm, 1993]) in non-cooperative environments.

In verifying the GPGP family of algorithms, we first showed that they duplicate and subsume the behaviors of the PGP algorithm. We then looked at several other issues:

General Performance: We examined the general performance of the most complex (all mechanisms in place) and least complex (all mechanisms off) members of the GPGP family in comparison to each other, and in comparison to a centralized scheduler reference implementation (as an upper bound). We looked at performance measures such as the total final quality achieved by the system, the amount of work done, the number of deadlines missed, and the termination time. The centralized schedule reference system is not an appropriate solution to the general coordination problem, even for cooperative groups of agents, for several reasons:

- The centralized scheduling agent becomes a possible single point of failure that can cause the entire system to fail (unlike the decentralized GPGP system).
- The centralized scheduling agent requires a complete, global view of the episode—a view that we mentioned earlier is not always easy to achieve. We do not account for any costs in building such a global view in the reference implementation (viewing it as an upper bound on performance). We do not allow dynamic changes in the episodic task structure (which might require rescheduling).
- The centralized reference scheduler uses an *optimal* single-agent schedule as a starting point. Since the problem of scheduling actions in even fairly simple task structures is NP-complete, the optimal scheduler's performance grows exponentially worse with the number of methods to be scheduled. Since the centralized reference scheduler has a global view and schedules all actions at all agents, the size of the centralized problem always grows faster than the size of the scheduling problems at GPGP agents with only partial views. The size of the episodes was kept small so that the centralized reference scheduler could find an optimal schedule in a reasonable amount of run time.

The performance of set of agents using all of the currently defined GPGP coordination mechanisms is good in comparison to the centralized reference system—GPGP agents produce on average 85% of the quality of the centralized upper bound reference solution, and do not miss any more deadlines.

Adding a Mechanism: We demonstrated that the addition of a particular mechanism can improve the system performance.

Family Design Space: We demonstrated the range of performance exhibited by different members of the GPGP algorithm family, obtained by simple parameterization of the individual coordination mechanisms.

Different Environments: We showed that different environments require different family members.

Load Balancing: We demonstrated how a new sixth mechanism, a load balancing mechanism, can be defined and integrated. We used this mechanism to show that the costs of using this mechanism are better balanced by performance improvements precisely when there is a large variance in the amount of work each agent would do by default. This result agrees with similar results in the distributed processing community on decentralized load balancing [Need a citation here].

Computational Organization Design: We recreated a set of experiments done by Burton and Obel [Burton and Obel, 1984] that examined the effects of technical interdependencies and organizational structure on the performance. GPGP team-oriented coordination mechanisms were used to define the organizational (team) structure, and TÆMS task structures defined the problem (as opposed to Burton and Obel's linear programs). We reached the same conclusions as Burton and Obel (that both do have an effect), and argued that one future application for TÆMS is as a tool for computational organization design.

TÆMS, as a framework to represent coordination problems in a formal, domain-independent way, is unlike any existing representation that is focussed on coordination issues. As a problem representation, it is richer and more expressive than game theory or team theory representations. For example, a typical game or team theory problem statement is concerned with a single decision; a typical TÆMS objective problem solving episode represents the possible outcomes of many sequences of choices (interrelated with one another). TÆMS can represent a game theoretic problem, and we could boil down a single decision made by an agent faced with a TÆMS task structure into a game theoretic problem (if there were no uncertainty involved—see Chapter 2). Because TÆMS is more expressive, we can use it to operationalize some of the rich but informal concepts of organizational science (such as *decomposability* in Section 6.7). Another difference between TÆMS and traditional distributed computing task representations is that TÆMS indicates that not all tasks in an episode need to be done.

To put the second part of this dissertation in context, the analysis of a simple distributed sensor network presented here is the first formal quantitative analysis of a DSN or DAI system, other than Sen's analysis of a two-agent distributed meeting scheduling system (developed at the same time) [Sen and Durfee, 1994]. Our analysis of a DSN system answers several questions, and explains phenomena observed in the work with the Distributed Vehicle Monitoring Testbed (DVMT) such as why different algorithms perform differently in different situations. The work described here moves beyond anecdotal data to design rules for DSN systems.

GPGP, the last major contribution of this dissertation, extends Durfee's work on Partial Global Planning by being domain independent, adding time deadlines, allowing the agents to be more heterogeneous, requiring less communication, and allowing communication at

multiple levels of detail. GPGP is a cooperative approach, and thus is different from algorithms that assume the agents act in rational self-interest only. For example, agents usually make decisions with much less *a priori* knowledge of the other agents' utilities than competitive game theory approaches. However, agents using GPGP mechanisms still make decisions in a boundedly rational way—choosing from among schedules in an attempt to maximize the system-wide utility given whatever subjective information they have.

7.4 Limitations of this work

The work described here is of course not without limitations. The framework for representing the task structure of any computational environment, /tems (Chapter 3), has several limitations. Some of these are currently being addressed in follow-up work. /tems/ describes an environment from three different viewpoints—the generative viewpoint of how individual episodes come about, the objective viewpoint of the actual structure of an episode, and the subjective viewpoint of the agents themselves. This dissertation concentrates on the objective viewpoint. It does not deal with the various representations of uncertainty that might be in an agent's subjective view—we are currently working on this aspect. The generative models in this dissertation are constructed from a statistical viewpoint—an episode can be characterized by the values of several random variables. While this is far better than using a set of single-instance examples, it may not be appropriate for dynamic environments, where the task structure itself is changing dramatically over time. More complex models of generation may be needed. For example, generation might be modeled by a set of meta-agents.

The idea of the task structure changing dramatically over time is also often important in modeling human organizations. Human organizations often face external forces that can reshape the very structure of their environment. For example, the way the government can enact new laws that affect certain businesses. Factors that influence organizational behavior or measures of organizational performance might change out from under the organization (which then must adapt). As we discussed in Section 2.2.2, these responses need not always be genuine, they might only be symbolic. Nothing in this dissertation deals with such immediate, unbounded uncertainty, or how and organization might decide between a symbolic or genuine response.[1] Furthermore, we have not discussed periodic or other cyclic, recurring task structures here, except as a source of future work.

Our results on the analysis of a simple Distributed Sensor Network in Chapter 4 are limited only by the simplicity of the domain. Some extensions, such as having more than one vehicle type, would be fairly simple. Other extensions, such as having the vehicles move in patterns, would invalidate parts of the model (but not all of it). For example, if vehicles tended to stay on certain paths (roads or underwater rifts), then this information could be used to give new *a priori* distributions of agent loads in an episode; the average load for each agent might very well be unequal. However, the work calculations in that chapter would still hold. The main limitation of the DSN analysis is that it cannot yet be automated.

The final major contribution, the Generalized Partial Global Planning family of algorithms (Chapter 5), has several limitations. First of all, the existing set of mechanisms can only be

[1] As a first approximation, one might adopt a multiple-criteria definition for quality, including an "external visibility" attribute. Symbolic responses (modeled as alternative methods of low cost, high external visibility, but low true quality) might then be chosen under stressful situations.

used by cooperative agents. Self-interested or antagonistic agents could take advantage of the commitment mechanism (for example) to cheat other agents. There is a great deal of work on mechanisms that can avoid this problem (at some cost of overhead). For example, commitments can be viewed as contracts and enforced by a third party. In some environments, repetitive episodes can bring about a rational self-interested decision to cooperate, such as Axelrod's tit-for-tat strategy in the Prisoner's Dilemma. The second limitation is that there are currently only 6 mechanisms, which will not suffice for arbitrary environments. The existing mechanisms do not, for example, allow for hierarchical organizations. Finally, there is as yet no learning component, so the designer is responsible for choosing when a mechanism should be used. We believe this limitation can also be overcome, as we will discuss in the section on Future Work (Section 7.5, next).

7.5 Future Work

The work described in this dissertation leads directly to several applications, extensions, and new directions.

The subjective level. We present the TÆMS' subjective level in this dissertation, but we do not use it to any great extent except in a few examples. Work has already begun on describing the various types of subjective uncertainties that can be present in task environments such as DRESUN. Previous DTT scheduling work has already considered some simple types of subjective uncertainty, such as the duration and actual quality of methods.

Other subjective level extensions include the indication of methods that provide only subjective information (i.e., provide better estimates for the duration of methods). The scheduler will then have to decide when to schedule these meta-level actions rather than domain-level method executions.

Resources. It is exciting to us that the basic TÆMS mechanisms can handle the representation of physical resources as described in Section 3.4.5. However, the notation is a bit ponderous and could be improved with some 'syntactic sugar' (as Steele would say). For example, since all resource relationships come in pairs, we could define them by a single double relationship (that would have the same semantics as the pair of relationships used now). More importantly, we can begin to develop or adapt local schedulers to schedule task structures with resources (both the Spring scheduler [Cheng *et al.*, 1988, Shen *et al.*, 1993] and Hildum's DSS scheduler [Hildum, 1994] are examples of such schedulers). Finally, we can begin to develop new coordination mechanisms to handle high-level, flexible resource usage. Some ideas about this can be found in Sycara's and Nieman's papers [Neiman *et al.*, 1994, Sycara *et al.*, 1991].

Hard real-time. We hint in Section 5.7 about using GPGP in concert with a true hard real-time operating system (i.e. Spring). Part of doing this involves the interface between the DTT scheduler and the Spring scheduler, part involves GPGP. TÆMS methods would probably be equivalent to Spring task groups. The resulting larger system would not be hard real-time, but only time-cognizant. It would be interesting to look at what a Spring guarantee means at a distance (i.e., wrapped in a GPGP commitment). An example of such a system might be a football team (with soft-real-time interaction between plays and hard real-time during the plays).

Load balancing. The demonstration in Section 6.6 hardly does this subject justice. A considerable amount of effort could be put into developing a more comprehensive set of GPGP load balancing mechanisms. The analysis of such mechanisms could be attempted by simplifying the environment as we did in Chapter 4 and by making assumptions about certain properties of the local scheduler with respect to termination.

Learning. As we described at the start and end of Chapter 5, GPGP is a family of algorithms—each of the five mechanisms described here are parameterized independently (the first two have three possible settings and the last three can be in or out) for a total of 72 combinations. More mechanisms such as the load balancing mechanism can been added to expand the family, but the family can also be enlarged by making each mechanism more situation-specific. For example, mechanisms can have their parameters set by a mapping from dynamic meta-level measurements such as an agent's load or the amount of real-time pressure. Mechanisms can be 'in' or 'out' for individual *classes* of task groups, or tasks, or even specific coordination relationships, that re-occur in particular environments. The cross product of these dynamic environmental cues provides a large but easily enumerated space of potential coordination responses that are amenable to the adaptation of the coordination mechanisms over time by standard machine learning techniques or case-based reasoning approaches.

GPGP as DAI infrastructure. GPGP could be developed as a modular, extendible, portable set of coordination modules that could be linked into existing applications to provide coordination assistance. The application would supply the task structure and code stubs for resolving and discovering coordination relationships, and possibly also a local scheduler. Mechanisms could be chosen by the programmer or by learning mechanisms such as those just described.

Societies creating agents. Throughout this dissertation, we have focused on the environment affecting agents. In reality, the environment and agents co-create one another (e.g., [Latour, 1987]). The only AI work ever to even being to address this question is Gasser and Ishida's work on self-organizing rule-based systems [Gasser and Ishida, 1991]. Gasser has been talking for years about the chicken-and-egg problem of agents and the society (e.g., [Gasser, 1991]), but most computer scientists cannot conceive of anything except societies created out of their agents, and never visa versa. The Gasser and Ishida paper presents a model where the problem exists and creates the agents—reversing the normal individualistic process. If there is a problem with this work, it is that the environment is only a simple set of rules. A clear forward direction is to use TÆMS task structures instead of rules to specify the task environment. What kind of rules would be needed for the formation of individuals (agents) from this beginning? When would individuals be reabsorbed? Would different environments cause the creation of different organizational structures? We assume that they would. This is related to the next point.

Computational organization design (COD). Future work will include the analysis of more traditional hierarchical organization forms. The addition of the concept of 'organizational role' can be accomplished through the use of non-local commitments (and their accompanying expectations). In its simplest sense, agent A's *organizational role* is a set of continuing non-local commitments to certain classes of tasks. By making continuous

commitments agents can avoid communicating about every episode anew—assuming the future structures are somewhat predictable. This reasoning leads naturally to learning organizational roles and to open systems concepts of temporarily settled questions (the current set of continuous commitments) and algorithms to re-open and re-settle these questions as the task environment changes. The development of such algorithms can then lead to exploring conceptions of agents as nothing more than convenient bundles of organizational roles [Gerson, 1976, Gasser *et al.*, 1989], and to when and why organizations do not act like 'big agents' [Allison, 1971]. We will also allow the expansion of commitments to meta-level roles (i.e., commitments to the coordination parameters in effect for certain classes of tasks). Related applications fall under the popular buzzwords Business Process Re-Engineering.

Computational organization theory. Related to COD is the idea of producing computational versions of traditional organization theories. For example, Masuch and Huang are formalizing J.D. Thompson's *Organizations in Action* using a multi-agent action logic [Masuch and Huang, 1994]. The book *Computational Organization Theory* [Carley and Prietula, 1994] is all about mathematical, logical, and computational approaches; a new book by the same editors will be about purely computational models (including the one presented in this dissertation). K. Crowston [personal communication] has talked about producing an entire book of computational versions of famous organizational theories, complete with the programs, for students to experiment with. Williamson's Transaction Costs Economics [Williamson, 1975] would be an example theory that could be modeled in TÆMS.

Negotiating Agents. A direct consequence of heterogeneous, dynamic, and real-time agents is the need for negotiation to solve conflicts. Even with a known global decision evaluation function, conflicting decisions of equal global value may have very different local value to the agents. Often the character of an early partial solution will have an impact on what style of coordination is needed. For example, if early partial results show poor data and low beliefs, the coordination mechanism may want to encourage redundant derivations of results in areas shared by more than one agent, or the parallel derivation of a result by two agents using different algorithms. The Pilot's Associate scenario in Section 3.6.3.4 occurs in too short a time-frame to allow negotiation between the agents, but other Pilot's Associate scenarios might profitably use negotiation techniques[2].

The PGP mechanism uses a shared global plan evaluation function that is parameterized. One extension is to allow the parameters (such as *redundancy* and *reliability*) to vary during problem solving. A negotiation facility could be developed to allow agents to usefully alter the global (or perhaps only semi-local — we are interested in agents that may develop only a partial view of what other agents are working on) decision criteria. Where the PGP mechanisms exchanged all local information, our extensions would allow for a multi-stage process [Conry *et al.*, 1991] where agents would communicate only the information believed relevant to the issue at hand. Agents could ask for more contextual

[2] For example, a sensor may overheat and be shut down by the system status module, even though it is a projected resource requirement for some tactical situation. The tactical planner and system status may negotiate over the amount of time that the damaged sensor can be used if the situation arises.

information when it is needed to resolve a conflict between agents. Agents would not automatically acquire information from other agents performing non-related problem solving activities. Negotiation can interact with sophisticated real-time local schedulers that have the capability to analyze potential future schedules.

GPGP in multi-agent systems. This is a whole area of mechanisms that could be incorporated from the rational, selfish agent community. New mechanisms would need to be developed to deal with the possibility of lying or other potentially 'uncooperative' behavior. Results from the game theoretic DAI community could be adapted fairly easily.

Dynamic Episodes. Up to this point, we have focused on the interrelationships and the uncertainty arising directly from the generative model of the environment, but we would also like to explore the uncertainty and variance arising from the difference in the objective and subjective models. Different relationships between these models will lead to different organizational structures. For example, in the case of uncertainty arising from the generative model of an environment, we showed that the wide variance in performance of a system of agents with static organizations in different episodes led to the use of meta-level communication to reorganize the agents to adapt to the particular episode at hand [Decker and Lesser, 1993c]. We are also looking to expand the TÆMS conception of environments to encompass more dynamic situations: another important source of environmental uncertainty. For example, in the *Tower of Babel* problem [Ishida, 1992], agents try to pile numbered blocks into a tower in numerical order. When many agents try to solve this problem without centralized coordination, they tend to bottleneck around the tower itself, bumping into one another. Ishida solves this problem by dividing the agents into two groups: one to collect blocks and bring them near (but not too near) the tower, and one group to stack the tower. From our point of view, the task structure at low levels is changing rapidly, and no agent can (or needs to) keep track of all the changes in relationships between agent and block positions. The rapid change in the environment means that agents' subjective views are always out of date. By dividing the agents into two groups, the bottleneck resource is controlled by organization—removing it from the uncertainties facing the agents. Stacking agents know there are not enough of them to saturate the bottleneck, and other agents no longer use the resource at all. The uncertainty is still there, but it no longer has an impact on the agents' decision-making process.

Coordination assistance for mixed human and computational agent systems. The operational vision of the ARPA/Rome Labs Planning Initiative (ARPI) is one of a *network* of workstations that enable *concurrent* and *distributed* assessment, plan generation, and analysis. It envisions that *groups* of humans and computational 'agents' will *cooperatively* create, resource, evaluate, execute, and monitor crisis management or manufacturing plans. Both problems and *opportunities* arise from a truly distributed and cooperative environment. A possible area of future work would be the development of support tools for distributed, cooperative work by groups of human and computational agents, based on TÆMS and GPGP.

In ARPA's Air Campaign Planning domain, for example, the development of objectives, target analyses, and situational analyses (e.g. identification of centers-of-gravity (COGs))

are mixed initiative processes that can take place concurrently, and are carried out by the JFACC (Joint Forces Air Component Commander) and his staff (intelligence, operations, logistics, weather, etc.). As the planner makes decisions to focus on certain enemy vulnerabilities, he creates (concurrent) taskings for intelligence and other staff members or computational agents; as the staff uncovers certain information, that same information may impact back on the main planning task (revealing logistical, intelligence, or other problems with certain targets). This pattern of *interrelated task structures* that are partially shared by multiple agents with mixed initiative tasking appears in many military and commercial environments (transportation planning, target assignment, 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).

In general, each person and computational agent involved in such distributed cooperative work has a set of initially unrelated tasks before them that currently need to be done (identifying targets in several areas, collecting intelligence on each of several disparate targets, etc.). One possible way to assist users in task selection is by developing something like 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. 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. A similar Agent Coordination Module (ACM) could be developed for purely computational agents. The ACM would provide agenda management services for coordinating computational agents in the context of user task-order preferences.

Such a set of tools will allow the development of systems that can truly support command and control planning. Such planning will involve both people at multiple workstations and computational agent assistants. The tools will help to manage and organize the workloads of both these types of 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. More information than simply task assignments and task coordination relationships is useful to communicate. 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 environments with different task coordination relationships).

Coordination and scheduling assistance for ARCADIA, a software engineering environment. Arguments similar to those in the last item apply to other domains, such as the concurrent engineering of products where designers, engineers, sales/marketing representatives and manufacturing staffers work together to develop a new product or an extension of an old product. We have already been exploring the use of our technology in the representation and management of large software engineering projects to help schedule tasks being performed by many different programmers using many different resources. Part of the ARCADIA [Kadia, 1992, Taylor *et al.*, 1988] environment is directly concerned with representing and tracking the state of: software development processes (including interrelationships), the products being produced, and the resources available. This information can be used by a UCAA-like application that can provide scheduling and coordination facilities to ARCADIA users. The most unique feature of this domain is the presence of high levels of uncertainty in task durations (compared to manufacturing domains).

Multi-agent distributed information gathering. Distributed information gathering applications using the Internet might look something like the following using the technology developed in this thesis: The original user query would be transformed into set of agents, each with their own plans for gathering information, where some plan elements must deal with the coordination of activities and construction of the final query response. Agents could be assigned to concurrently pursue different sources to answer different aspects of the query or to make use of alternative types of sources (e.g., text vs. images) to generate a more comprehensive answer. The agents would proceed to work in a distributed, asynchronous fashion, but there may need to be coordination among the agents. For example, the results of work by one agent may suggest the need for some of the existing agents to gather additional information from their sources or use alternative sources, or it might suggest the need for additional agents. The agents must also deal with uncertainties about the availability of sources and the workload associated with the sources, which may require a new division of tasks among the agents.

REFERENCES

[Adler *et al.*, 1989] M.R. Adler, A.B. Davis, R. Weihmayer, and R.W. Worrest. Conflict resolution strategies for nonhierarchical distributed agents. In L. Gasser and M. N. Huhns, editors, *Distributed Artificial Intelligence, Vol. II*, pages 139–162. Pitman Publishing Ltd., 1989. Also COINS Technical Report 88-89, University of Massachusetts, 1988.

[Agre, 1991] Philip E. Agre. Title undecided. To be published by Cambridge University Press. Position paper presented at AAAI Fall Symposium on Knowledge and Action at Social and Organizational Levels, 1991.

[Allison, 1971] Graham T. Allison. *Essence of Decision: Explaining the Cuban Missle Crisis*. Little, Brown and Company, Boston, 1971.

[Anderton, 1994] D.L. Anderton. Anderton questions for decker's defense. Private Communication., 1994.

[Axelrod, 1984] R. Axelrod. *The Evolution of Cooperation*. Basic Books, New York, 1984.

[Blau and Schwartz, 1984] P.M. Blau and J.E. Schwartz. *Crosscutting Social Circles: Testing a Macrostructural Theory of Group Relationships*. Academic Press, New York, 1984.

[Boddy and Dean, 1989] Mark Boddy and Thomas Dean. Solving time-dependent planning problems. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 979–984, Detroit, August 1989.

[Bond and Gasser, 1988] Alan H. Bond and Les Gasser. An analysis of problems and research in DAI. In Alan H. Bond and Les Gasser, editors, *Readings in Distributed Artificial Intelligence*, pages 3–35. Morgan Kaufmann, 1988.

[Bonissone and Decker, 1986] Piero P. Bonissone and Keith S. Decker. Selecting uncertainty calculi and granularity: An experiment in trading-off precision and complexity. In L. N. Kanal and J. F. Lemmer, editors, *Uncertainty in Artificial Intelligence*. North Holland, 1986.

[Broverman *et al.*, 1987] C. A. Broverman, K. E. Huff, and V. R. Lesser. The role of plan recognition in design of an intelligent user interface. In *Proceedings of the IEEE Systems, Man, and Cybernetics Conference*, pages 863–868, Atlanta, Georgia, 1987.

[Burrell *et al.*, 1994] G. Burrell, M. Reed, M. Alverson, M. Calás, and L. Smircich. Why Organization? why now? *Organization*, 1(1):5–17, 1994.

[Burt, 1982] R.S. Burt. *Toward a structural theory of action : network models of social structure, perception, and action*. Quantitative studies in social relations. Academic Press, New York, 1982.

[Burton and Obel, 1984] Richard M. Burton and Børge Obel. *Designing Efficient Organizations: Modelling and Experimentation*. North Holland, Amsterdam, 1984.

[Cammarata *et al.*, 1983] Stephanie Cammarata, David McArthur, and Randall Steeb. Strategies of cooperation in distributed problem solving. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 767–770, Karlsruhe, Germany, August 1983.

[Carley and Prietula, 1994] K.M. Carley and M.J. Prietula, editors. *Computational Organization Theory*. Lawrence Erlbaum Associates, 1994.

[Carver *et al.*, 1991] Norman Carver, Zarko Cvetanovic, and Victor Lesser. Sophisticated cooperation in FA/C distributed problem solving systems. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 191–198, Anaheim, July 1991.

[Carver, 1994] N. Carver. A note on multi-agent information retrieval. Private Communication., 1994.

[Castelfranchi, 1993] 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.

[Chambers *et al.*, 1984] F.B. Chambers, D.A. Duce, and G.P. Jones, editors. *Distributed Computing*, volume 20 of *APIC Studies in Data Processing*. Academic Press, London, 1984.

[Chandrasekaran, 1981] B. Chandrasekaran. Natural and social system metaphors for distributed problem solving: Introduction to the issue. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(1):1–5, January 1981.

[Cheng *et al.*, 1986] S. Cheng, J.A. Stankovic, and K. Ramamritham. Dynamic scheduling of groups of tasks with precedence constraints in distributed hard real-time systems. In *Real-time Systems Symposium*, December 1986.

[Cheng *et al.*, 1988] S. Cheng, J. Stankovic, and K. Ramamritham. Scheduling algorithms for hard real-time systems. In *Hard Real-Time Systems*. IEEE Press, 1988.

[Chisholm, 1989] Donald Chisholm. *Coordination Without Hierarchy: Informal Structures in Multiorganizational Systems*. University of California Press, Berkeley, 1989.

[Cohen and Levesque, 1990] Philip R. Cohen and Hector J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42(3):213–261, 1990.

[Cohen *et al.*, 1989] Paul Cohen, Michael Greenberg, David Hart, and Adele Howe. Trial by fire: Understanding the design requirements for agents in complex environments. *AI Magazine*, 10(3):33–48, Fall 1989. Also COINS-TR-89-61.

[Cohen, 1991] Paul R. Cohen. A survey of the Eighth National Conference on Artificial Intelligence: Pulling together or pulling apart? *AI Magazine*, 12(1):16–41, Spring 1991.

[Cohen, 1992] Paul R. Cohen. Tactics for generalization. Course notes for Experimental Design in AI, 1992.

[Conry *et al.*, 1991] S. E. Conry, K. Kuwabara, V. R. Lesser, and R. A. Meyer. Multistage negotiation for distributed constraint satisfaction. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6), November 1991.

[Corkill and Lesser, 1983] Daniel D. Corkill and Victor R. Lesser. The use of meta-level control for coordination in a distributed problem solving network. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 748–755, Karlsruhe, Germany, August 1983.

[Corkill, 1979] Daniel D. Corkill. Hierarchical planning in a distributed environment. In *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, pages 168–175, Cambridge, 1979. Also COINS TR–79–13.

[Crowston, 1994] Kevin Crowston. Coordinating restaurant service: An analysis of TGI Friday's process. Presentation given at the TIMS/ORSA Workshop on Mathematical and Computational Organization Theory, April 1994. Crowston is a professor at the University of Michigan School of Business.

[Daniel, 1978] W. W. Daniel. *Applied Nonparametric Statistics*. Houghton-Mifflin, Boston, 1978.

[Davis and Smith, 1983] R. Davis and R. G. Smith. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, 20(1):63–109, January 1983.

[Dean and Wellman, 1991] T. L. Dean and M. P Wellman. *Planning and Control*. Morgan Kaufman Publishers, San Mateo, California, 1991.

[Dean, 1987] Thomas Dean. Planning, execution, and control. In *Proceedings of the DARPA Knowledge-based Planning Workshop*, December 1987.

[Decker and Lesser, 1990] Keith S. Decker and Victor R. Lesser. Extending the partial global planning framework for cooperative distributed problem solving network control. In *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 396–408, San Diego, November 1990. Morgan Kaufmann. Also COINS TR-90-81.

[Decker and Lesser, 1992] Keith S. Decker and Victor R. Lesser. Generalizing the partial global planning algorithm. *International Journal of Intelligent and Cooperative Information Systems*, 1(2):319–346, June 1992.

[Decker and Lesser, 1993a] Keith S. Decker and Victor R. Lesser. Analyzing a quantitative coordination relationship. *Group Decision and Negotiation*, 2(3):195–217, 1993.

[Decker and Lesser, 1993b] 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.

[Decker and Lesser, 1993c] Keith S. Decker and Victor R. Lesser. A one-shot dynamic coordination algorithm for distributed sensor networks. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 210–216, Washington, July 1993.

[Decker and Lesser, 1993d] 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.

[Decker and Lesser, 1993e] Keith S. Decker and Victor R. Lesser. Quantitative modeling of complex environments. *International Journal of Intelligent Systems in Accounting, Finance, and Management*, 2(4):215–234, December 1993. Special issue on "Mathematical and Computational Models of Organizations: Models and Characteristics of Agent Behavior".

[Decker *et al.*, 1990] Keith S. Decker, Victor R. Lesser, and Robert C. Whitehair. Extending a blackboard architecture for approximate processing. *The Journal of Real-Time Systems*, 2(1/2):47–79, 1990.

[Decker *et al.*, 1991] Keith S. Decker, Alan J. Garvey, Marty A. Humphrey, and Victor R. Lesser. Effects of parallelism on blackboard system scheduling. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pages 15–21, Sydney, Australia, August 1991. Extended version to appear in the International Journal of Pattern Recognition and Artificial Intelligence 7(2) 1993.

[Decker *et al.*, 1992] Keith Decker, Alan Garvey, Marty Humphrey, and Victor Lesser. A blackboard system for real-time control of approximate processing. In *Proceedings of the 25th Hawaii International Conference on System Sciences*, January 1992. Extended version to appear in the International Journal of Pattern Recognition and Artificial Intelligence 7(2) 1993.

[Decker *et al.*, 1993a] Keith S. Decker, Alan J. Garvey, Marty A. Humphrey, and Victor R. Lesser. Control heuristics for scheduling in a parallel blackboard system. *International Journal of Pattern Recognition and Artificial Intelligence*, 7(2):243–264, 1993.

[Decker *et al.*, 1993b] Keith S. Decker, Alan J. Garvey, Marty A. Humphrey, and Victor R. Lesser. A real-time control architecture for an approximate processing blackboard system. *International Journal of Pattern Recognition and Artificial Intelligence*, 7(2):265–284, 1993.

[Decker, 1987] Keith S. Decker. Distributed problem solving: A survey. *IEEE Transactions on Systems, Man, and Cybernetics*, 17(5):729–740, September 1987.

[Decker, 1994a] Keith S. Decker. Distributed artificial intelligence testbeds. In G. O'Hare and N. Jennings, editors, *Foundations of Distributed Artificial Intelligence*, chapter 3. Wiley Inter-Science, 1994. Forthcoming.

[Decker, 1994b] Keith S. Decker. *Environment Centered Analysis and Design of Coordination Mechanisms*. PhD thesis, University of Massachusetts, 1994.

[Decker, 1994c] Keith S. Decker. Tæms: A framework for analysis and design of coordination mechanisms. In G. O'Hare and N. Jennings, editors, *Foundations of Distributed Artificial Intelligence*, chapter 16. Wiley Inter-Science, 1994. Forthcoming.

[Dubois and Prade, 1984] D. Dubois and H. Prade. Criteria aggregation and ranking of alternatives in the framework of fuzzy set theory. In H.J. Zimmermen, L. A. Zadeh, and B.R. Gains, editors, *TIMS/Studies in Management Science*, volume 20, pages 209–240. Elsevier Science Publishers, 1984.

[Durfee and Lesser, 1987] Edmund H. Durfee and Victor R. Lesser. Using partial global plans to coordinate distributed problem solvers. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, August 1987.

[Durfee and Lesser, 1988a] E. Durfee and V. Lesser. Predictability vs. responsiveness: Coordinating problem solvers in dynamic domains. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 66–71, St. Paul, Minnesota, August 1988.

[Durfee and Lesser, 1988b] Edmund H. Durfee and Victor R. Lesser. Incremental planning to control a time-constrained, blackboard-based problem solver. *IEEE Transactions on Aerospace and Electronic Systems*, 24(5):647–662, September 1988.

[Durfee and Lesser, 1989] Edmund H. Durfee and Victor R. Lesser. Negotiating task decomposition and allocation using partial global planning. In L. Gasser and M. N. Huhns, editors, *Distributed Artificial Intelligence, Vol. II*. Pitman Publishing Ltd., 1989.

[Durfee and Lesser, 1991] 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.

[Durfee and Montgomery, 1990] Edmund H. Durfee and Thomas A. Montgomery. A hierarchical protocol for coordinating multiagent behaviors. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, July 1990.

[Durfee and Montgomery, 1991] 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.

[Durfee *et al.*, 1987] 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.

[Durfee *et al.*, 1989] E. H. Durfee, V. R. Lesser, and D. D. Corkill. Cooperative distributed problem solving. In A. B. Barr, P. Cohen, and E. Feigenbaum, editors, *The Handbook of Artificial Intelligence*, volume 4, pages 83–147. Addison Wesley, 1989.

[Durfee *et al.*, 1993] E.H. Durfee, J. Lee, and P.J. Gmytrasiewicz. Overeager reciprocal rationality and mixed strategy equilibria. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 225–230, Washington D.C., 1993.

[Durfee, 1987] Edmund H. Durfee. *A Unified Approach to Dynamic Coordination: Planning Actions and Interactions in a Distributed Problem Solving Network*. PhD thesis, University of Massachusetts, 1987. Also COINS-TR-87-84.

[Ephrati and Rosenschein, 1994] E. Ephrati and J.S. Rosenschein. Divide and conquer in multi-agent planning. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 375–380, Seattle, 1994. AAAI Press/MIT Press.

[Fox, 1981] Mark S. Fox. An organizational view of distributed systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(1):70–80, January 1981.

[Galbraith, 1977] J. Galbraith. *Organizational Design*. Addison-Wesley, Reading, MA, 1977.

[Garvey and Lesser, 1993] Alan Garvey and Victor Lesser. Design-to-time real-time scheduling. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(6):1491–1502, 1993.

[Garvey and Lesser, 1994] Alan Garvey and Victor Lesser. A survey of research in deliberative real-time artificial intelligence. *The Journal of Real-Time Systems*, 6, 1994.

[Garvey *et al.*, 1993] 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.

[Garvey *et al.*, 1994] Alan Garvey, Keith Decker, and Victor Lesser. A negotiation-based interface between a real-time scheduler and a decision-maker. CS Technical Report 94–08, University of Massachusetts, 1994.

[Gasser and Ishida, 1991] Les Gasser and Toru Ishida. A dynamic organizational architecture for adaptive problem solving. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 185–190, Anaheim, July 1991.

[Gasser *et al.*, 1989] Les Gasser, N. F. Rouquette, R. W. Hill, and J. Lieb. Representing and using organizational knowledge in distributed AI systems. In L. Gasser and M. N. Huhns, editors, *Distributed Artificial Intelligence, Vol. II*. Pitman Publishing Ltd., 1989.

[Gasser, 1991] Les Gasser. Social conceptions of knowledge and action. *Artificial Intelligence*, 47(1):107–138, 1991.

[Genesereth *et al.*, 1986] M. R. Genesereth, M. L. Ginsberg, and J. S. Rosenschein. Cooperation without communication. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 51–57, Philadelphia, PA., August 1986.

[Georgeff, 1983] Michael Georgeff. Communication and interaction in multiagent planning. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 125–129, August 1983.

[Georgeff, 1984] Michael Georgeff. A theory of action for multiagent planning. In *Proceedings of the Fourth National Conference on Artificial Intelligence*, pages 121–125, August 1984.

[Gerson, 1976] Elihu M. Gerson. On 'quality of life'. *American Sociological Review*, 41:793–806, 1976.

[Gmytrasiewicz *et al.*, 1991] Piotr J. Gmytrasiewicz, Edmund H. Durfee, and David K. Wehe. A decision-theoretic approach to coordinating multiagent interactions. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pages 62–68, Sydney, Australia, August 1991.

[Goldman and Rosenschein, 1993] C. Goldman and J. R. Rosenschein. Emergent coordination through the use of cooperative state-changing rules. In *Proceedings of the Twelfth International Workshop on Distributed AI*, Hidden Valley, PA, May 1993.

[Halpern and Moses, 1985] Joseph Y. Halpern and Yoram Moses. A guide to the modal logics of knowledge and belief: Preliminary draft. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 480–490, August 1985.

[Halpern, 1986] Joseph Y. Halpern, editor. *Theoretical Aspects of Reasoning About Knowledge: Proceedings of the 1986 Conference*. Morgan Kaufmann Publishers, Inc., March 1986.

[Hayes-Roth *et al.*, 1988] F. Hayes-Roth, L. Erman, S. Fouse, J. Lark, and J. Davidson. ABE: A cooperative operating system and development environment. In Alan H. Bond and Les Gasser, editors, *Readings in Distributed Artificial Intelligence*, pages 457–490. Morgan Kaufmann, 1988.

[Hewitt, 1986] Carl Hewitt. Offices are open systems. *ACM Transactions on Office Information Systems*, 4(3):271–287, July 1986.

[Hewitt, 1991] Carl Hewitt. Open information systems semantics for distributed artificial intelligence. *Artificial Intelligence*, 47(1):79–106, 1991.

[Hildum, 1994] David W. Hildum. *Flexibility in a Knowledge-Based System for Solving Dynamic Resource-Constrained Scheduling Problems*. PhD thesis, Department of Computer Science, University of Massachusetts, Amherst, September 1994.

[Ho, 1980] Y.-C. Ho. Team decsision theory and information structures. *Proceedings of the IEEE*, 68, June 1980.

[Horvitz, 1988] Eric J. Horvitz. Reasoning under varying and uncertain resource constraints. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, August 1988.

[Huff and Lesser, 1988] K. E. Huff and V. R. Lesser. A plan-based intelligent assistant that supports the software development process. In *Proceedings of the Third ACM Symposium on Software Development Environments*, Boston, MA, November 1988.

[Hulthage, 1994] Ingemar Hulthage, editor. *Computational Organization Design*. AAAI Spring Symposium, 1994. Working Notes.

[Ishida, 1992] Toru Ishida. Tower of Babel: Towards organization-centered problem solving. In *Proceedings of the 11th Workshop on Distributed Artificial Intelligence*, pages 141–153, The Homestead, Michigan, 1992.

[Jennings, 1993] N. R. Jennings. Commitments and conventions: The foundation of coordination in multi-agent systems. *The Knowledge Engineering Review*, 8(3):223–250, 1993.

[Kadane and Larkey, 1982] J.B. Kadane and P.D. Larkey. Subjective probability and the theory of games. *Management Science*, 28(2):113–120, February 1982.

[Kadia, 1992] R. Kadia. Issues encountered in building a flexible software development environment: Lessons from the Arcadia project. In *Proceedings of the Fifth ACM SIGSOFT Symposium on Software Development Environments (SDE5)*, pages 169–180, Tyson's Corner, VA, December 1992.

[Kleijnen, 1987] Jack P. C. Kleijnen. *Statistical Tools for Simulation Practitioners*. Marcel Dekker, New York, 1987.

[Knoblock, 1991] Craig A. Knoblock. Search reduction in hierarchical problem solving. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, Anaheim, July 1991.

[Kornfeld and Hewitt, 1981] William A. Kornfeld and Carl E. Hewitt. The scientific community metaphor. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(1):24–33, January 1981.

[Lampson *et al.*, 1981] B. W. Lampson, M. Paul, and H. J. Siegert, editors. *Distributed Systems Architecture and Implementation*. Springer-Verlag, New York, 1981.

[Lander and Lesser, 1989] Susan Lander and Victor R. Lesser. A framework for the integration of cooperative knowledge-based systems. In *Proceedings of the 4th IEEE International Symposium on Intelligent Control*, pages 472–477, September 1989.

[Latour, 1987] Bruno Latour. *Science in Action*. Harvard University Press, Cambridge, MA, 1987.

[Lawrence and Lorsch, 1967] Paul Lawrence and Jay Lorsch. *Organization and Environment*. Harvard University Press, Cambridge, MA, 1967.

[Lesser and Corkill, 1981] Victor R. Lesser and Daniel D. Corkill. Functionally accurate, cooperative distributed systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(1):81–96, January 1981.

[Lesser and Corkill, 1983] Victor R. Lesser and Daniel D. Corkill. The distributed vehicle monitoring testbed. *AI Magazine*, 4(3):63–109, Fall 1983.

[Lesser *et al.*, 1987] V. R. Lesser, D. D. Corkill, and E. H. Durfee. An update on the distributed vehicle monitoring testbed. Computer Science Technical Report 87–111, University of Massachusetts, 1987.

[Lesser *et al.*, 1988] Victor R. Lesser, Jasmina Pavlin, and Edmund Durfee. Approximate processing in real-time problem solving. *AI Magazine*, 9(1):49–61, Spring 1988.

[Lesser, 1991] V. R. Lesser. A retrospective view of FA/C distributed problem solving. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6):1347–1363, November 1991.

[Levesque *et al.*, 1990] Hector J. Levesque, Philip R. Cohen, and José H. T. Nunes. On acting together. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 94–99, July 1990.

[Levitt *et al.*, 1994] R.E. Levitt, P.G. Cohen, J.C. Kunz, C. Nass, T. Christiansen, and Y. Jin. The virtual design team: Simulating how organizational structure and communication tools affect team performance. In K.M. Carley and M.J. Prietula, editors, *Computational Organization Theory*. Lawrence Erlbaum Associates, 1994.

[Lin and Carley, 1993] Z. Lin and K. Carley. Proactive or reactive: An analysis of the effect of agent style on organizational decision-making performance. *International Journal of Intelligent Systems in Accounting, Finance, and Management*, 2(4):271–289, December 1993. Special issue on "Mathematical and Computational Models of Organizations: Models and Characteristics of Agent Behavior".

[Liu *et al.*, 1991] J. W. S. Liu, K. J. Lin, W. K. Shih, A. C. Yu, J. Y. Chung, and W. Zhao. Algorithms for scheduling imprecise computations. *IEEE Computer*, 24(5):58–68, May 1991.

[Luce and Raiffa, 1958] R. Duncan Luce and Howard Raiffa. *Games and Decisions: Introduction and Critical Survey*. John Wiley & Sons, New York, 1958.

[Malone and Crowston, 1991] Thomas W. Malone and Kevin Crowston. Toward an interdisciplinary theory of coordination. Center for Coordination Science Technical Report 120, MIT Sloan School of Management, 1991.

[Malone *et al.*, 1983] T. W. Malone, R. E. Fikes, and M. T. Howard. Enterprise: A market-like task scheduler for distributed computing environments. Technical Report CISR-WP-111, MIT Center for Information Systems Research, October 1983.

[Malone *et al.*, 1986] T. W. Malone, R. E. Fikes, K. R. Grant, and M. T. Howard. Market-like task scheduling in distributed computing environments. Technical Report CISR-WP-139, MIT Center for Information Systems Research, March 1986.

[Malone *et al.*, 1993] Thomas W. Malone, Kevin Crowston, Jintae Lee, and Brian Pentland. Tools for inventing organizations: Toward a handbook of organizational processes. Center for Coordination Science Technical Report 141, MIT Sloan School of Management, 1993.

[Malone, 1987] Thomas W. Malone. Modeling coordination in organizations and markets. *Management Science*, 33:1317–1332, 1987.

[Malone, 1988] Thomas W. Malone. What is coordination theory? In *Proceedings of the National Science Foundation Coordination Theory Workshop*, February 1988.

[March and Simon, 1958] J. G. March and H. A. Simon. *Organizations*. Wiley, New York, 1958.

[Masuch and Huang, 1994] M. Masuch and Z. Huang. A logical deconstruction of organizational action: Formalizing J.D. thompson's *organizations in action* in a multi-agent action logic. CCSOM Woring Paper 94-120, Department of Statistics and Methodology, PSCW, University of Amsterdam, Netherlands, 1994.

[McKelvey, 1982] Bill McKelvey. *Organizational Systematics*. University of California Press, Berkeley, CA, 1982.

[Meyer and Rowan, 1977] J.W. Meyer and B. Rowan. Institutionalized organizations: Formal structures as myth and ceremony. *American Journal of Sociology*, 83:340–363, 1977.

[Meyer and Scott, 1983] J.W. Meyer and W.R. Scott. *Organizational Environments: Ritual and Rationality*. Sage Publications, Beverly Hills, CA, 1983.

[Mi and Scacchi, 1990] P. Mi and W. Scacchi. A knowledge-based environment for modeling and simulating software engineering processes. *IEEE Transactions on Knowledge and Data Engineering*, 2(3):283–294, September 1990.

[Mirchandaney *et al.*, 1989] R. Mirchandaney, D. Towsley, and J. Stankovic. Analysis of the effects of delays on load sharing. *IEEE Transactions on Computers*, 38(11):1513–1525, November 1989.

[Moe, 1984] Terry M. Moe. The new economics of organization. *American Journal of Political Science*, 28(4):739–777, November 1984.

[Moehlman *et al.*, 1992] T. Moehlman, V. Lesser, and B. Buteau. Decentralized negotiation: An approach to the distributed planning problem. *Group Decision and Negotiation*, 1(2):161–192, 1992.

[Montgomery and Durfee, 1992] Thomas A. Montgomery and Edmund H. Durfee. Search reduction in hierarchical distributed problem solving. In *Proceedings of the Eleventh International Workshop on Distributed AI*, Glen Arbor, Michigan, February 1992.

[Neiman *et al.*, 1994] D.E. Neiman, D.W. Hildum, V.R. Lesser, and T.W. Sandholm. Exploiting meta-level information in a distributed scheduling system. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Seattle, August 1994.

[Nilakant and Rao, 1994] V. Nilakant and H. Rao. Agency theory and uncertainty in organizations: An evaluation. *Organizational Studies*, 15(5):649–672, 1994.

[Nirenburg and Lesser, 1988] 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.

[Oates *et al.*, 1994] Tim Oates, M. V. Nagendra Prasad, and Victor R. Lesser. Cooperative information gathering: A distributed problem solving approach. Technical Report 94-66, Department of Computer Science, University of Massachusetts, September 1994.

[Ow *et al.*, 1989] 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.

[Padgett, 1992] John F. Padgett. The alchemist of contingency theory. *American Journal of Sociology*, 97(5):1462–1470, March 1992.

[Paker and Verjus, 1983] Y. Paker and J.-P. Verjus, editors. *Distributed Computing Systems: Synchronization, Control, and Communication*. Academic Press, London, 1983.

[Pavlin, 1983] Jasmina Pavlin. Predicting the performance of distributed knowledge-based systems: A modeling approach. In *Proceedings of the Third National Conference on Artificial Intelligence*, pages 314–319, Washington, D.C., August 1983.

[Perrow, 1986] Charles Perrow. *Complex Organizations*. Random House, New York, 1986.

[Pfeffer, 1983] J. Pfeffer. Organizational demography. In L.L. Cummings and B.M. Staw, editors, *Research in Organizational Behavior*, volume 5. JAI Press, Greenwich, CT, 1983.

[Pfeffer, 1991] J. Pfeffer. Oganizational theory and structural perspectives on management. *Journal of Management*, 17(4):789–803, December 1991.

[Pfeffer, 1993] J. Pfeffer. Barriers to the advance of organizational science: Paradigm development as an independent variable. *Academy of Management Review*, 18(4):599–620, 1993.

[Pollack and Ringuette, 1990] Martha E. Pollack and Marc Ringuette. Introducing Tileworld: Experimentally evaluating agent architectures. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 183–189, Boston, July 1990.

[Ramamritham *et al.*, 1990] Krithi Ramamritham, John A. Stankovic, and Perng-Fei Shiah. Efficient scheduling algorithms for real-time multiprocessor systems. *IEEE Transactions on Parallel and Distributed Systems*, 1(2):184–195, April 1990.

[Rapoport and Guyer, 1966] A. Rapoport and M. Guyer. A taxonomy of 2 x 2 games. In *Yearbook of the Society for General Systems Research XI*, pages 203–214. 1966.

[Rasmusen, 1989] Eric Rasmusen. *Games and Information: An Introduction to Game Theory*. Basil Blackwell, 1989.

[Reed and Lesser, 1980] S. Reed and V.R. Lesser. Division of labor in honey bees and distributed focus of attention. COINS Technical Report 80–17, University of Massachusetts, November 1980.

[Robinson and Fickas, 1990] William N. Robinson and Stephen Fickas. Negotiation freedoms for requirements engineering. Technical Report CIS-TR-90-04, Department of Computer and Information Science, University of Oregon, April 1990.

[Rosenschein and Breese, 1989] J. S. Rosenschein and J. S. Breese. Communication-free interactions among rational agents: A probabilistic approach. In L. Gasser and M. N. Huhns, editors, *Distributed Artificial Intelligence, Vol. II*. Pitman Publishing Ltd., 1989.

[Rosenschein and Genesereth, 1984] J. S. Rosenschein and M. R. Genesereth. Communication and cooperation. Technical Report HPP-84-5, Stanford Heuristic Programming Project, 1984.

[Rosenschein and Genesereth, 1985] J. S. Rosenschein and M. R. Genesereth. Deals among rational agents. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 91–99, August 1985.

[Russell and Wefald, 1991] Stuart Russell and Eric Wefald. *Do the Right Thing: Studies in Limited Rationality*. MIT Press, Cambridge, MA, 1991.

[Russell and Zilberstein, 1991] Stuart J. Russell and Shlomo Zilberstein. Composing real-time systems. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pages 212–217, Sydney, Australia, August 1991.

[Sandholm, 1993] Tuomas Sandholm. An implementation of the contract net protocol based on marginal cost calculations. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 256–262, Washington, July 1993.

[Scott, 1987] W. Richard Scott. *Organizations: Rational, Natural, and Open Systems*. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1987.

[Sen and Durfee, 1994] S. Sen and E. Durfee. On the design of an adaptive meeting scheduler. In *Proceedings IEEE Conference on AI Applications*, 1994.

[Shen *et al.*, 1993] Chia Shen, Krithi Ramamritham, and John A. Stankovic. Resource reclaiming in multiprocessor real-time systems. *IEEE Transactions on Parallel and Distributed Systems*, 4(4):382–398, April 1993.

[Shoham and Tennenholtz, 1992] Y. Shoham and M. Tennenholtz. On the synthesis of useful social laws for artificial agent societies (preliminary report). In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 276–281, San Jose, July 1992.

[Shoham, 1991] Yoav Shoham. AGENT0: A simple agent language and its interpreter. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 704–709, Anaheim, July 1991.

[Šiljak, 1991] D. D. Šiljak. *Decentralized Control of Complex Systems*. Academic Press, Inc., Boston, 1991.

[Simon, 1957] Herbert A. Simon. *Models of Man*. Wiley, New York, 1957.

[Simon, 1982] Herbert A. Simon. *Models of Bounded Rationality, Volume 2*. The MIT Press, Cambridge, MA, 1982.

[Smith and Broadwell, 1987] David Smith and Martin Broadwell. Plan coordination in support of expert systems integration. In *Proceedings of the DARPA Knowledge-Based Planning Workshop*, pages 12.1–12.6, December 1987.

[So and Durfee, 1992] Y.-P. So and E. H. Durfee. A distributed problem solving infrastructure for computer network management. *International Journal of Intelligent and Cooperative Information Systems*, 1(2):363–392, June 1992.

[Spangler *et al.*, 1978] E. Spangler, M.A. Gordon, and R.M. Pipkin. Token women: An empirical test of kanter's hypothesis. *American Journal of Sociology*, 85:160–170, 1978.

[Stankovic and Ramamritham, 1987] J. A. Stankovic and K. Ramamritham. The design of the spring kernel. In *Proceedings of the 1987 Real-time Systems Symposium*, December 1987.

[Stankovic *et al.*, 1989] J. A. Stankovic, K. Ramamritham, and D. Niehaus. On using the Spring kernel to support real-time AI applications. In *Proceedings of the EuroMicro Workshop on Real-time Systems*, 1989.

[Stankovic, 1984a] J. A. Stankovic. A perspective on distributed computing systems. *IEEE Transactions on Computers*, C–33(12):1102–1115, December 1984.

[Stankovic, 1984b] J. A. Stankovic. Simulations of three adaptive, decentralized controlled, job scheduling algorithms. *Computer Networks*, 8:199–217, 1984.

[Stankovic, 1985] J. A. Stankovic. An application of bayesian decision theory to decentralized control of job scheduling. *IEEE Transactions on Computers*, C-34(2):117–130, February 1985.

[Stinchcombe, 1987] Arthur L. Stinchcombe. *Constructing Social Theories*. University of Chicago Press, Chicago, 1987.

[Stinchcombe, 1990] Arthur L. Stinchcombe. *Information and Organizations*. University of California Press, Berkeley, CA, 1990.

[Sugawara and Lesser, 1993] Toshiharu Sugawara and Victor R. Lesser. On-line learning of coordination plans. Computer Science Technical Report 93–27, University of Massachusetts, 1993.

[Sycara *et al.*, 1991] K. Sycara, S. Roth, N. Sadeh, and M. Fox. Distributed constrained heuristic search. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6):1446–1461, November/December 1991.

[Taylor *et al.*, 1988] R.N. Taylor, F.C. Belz, L.A. Clarke, L. Osterweil, W.W. Selby, J.C. Wileden, A.L. Wolfe, and M. Young. Foundations for the Arcadia environment architecture. In *Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, pages 1–12, Boston, MA, November 1988.

[Thompson, 1967] James D. Thompson. *Organizations in Action*. McGraw-Hill, New York, 1967.

[v. Bochmann, 1983] Gregor v. Bochmann. *Distributed Systems Design*. Springer-Verlag, Berlin, 1983.

[v. Martial, 1990] Frank v. Martial. A conversation model for resolving conflicts among distributed office activities. In *Proceedings of the Fifth Conference on Office Information Systems*, pages 99–108, Cambridge, MA, April 1990.

[v. Martial, 1992] Frank v. Martial. *Coordinating Plans of Autonomous Agents*. Springer-Verlag, Berlin, 1992. Lecture Notes in Artificial Intelligence no. 610.

[Weber, 1947] Max Weber. *The Theory of Social and Economic Organization*. Oxford University Press, New York, 1947. Translated and edited by A. M. Henderson and T. Parsons.

[Wellman, 1993] Michael Wellman. A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research*, 1:1–23, 1993.

[Wesson *et al.*, 1981] Robert Wesson, Frederick Hayes-Roth, John W. Burge, Cathleen Statz, and Carl A. Sunshine. Network structures for distributed situation assessment. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(1):5–23, January 1981.

[Westbrook *et al.*, 1994] 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.

[Williamson, 1975] Oliver E. Williamson. *Markets and Hierarchies: Analysis and Antitrust Implications*. The Free Press, New York, 1975.

[Zhao *et al.*, 1987] W. Zhao, K. Ramamritham, and J. A. Stankovic. Scheduling tasks with resource requirements in hard real-time systems. *IEEE Transactions on Software Engineering*, May 1987.

[Zlotkin and Rosenschein, 1990] Gilad Zlotkin and Jeffrey S. Rosenschein. Blocks, lies, and postal freight: The nature of deception in negotiation. In *Proceedings of the Tenth International Workshop on Distributed AI*, Texas, October 1990.

[Zlotkin and Rosenschein, 1991] G. Zlotkin and J. S. Rosenschein. Incomplete information and deception in multi-agent negotiation. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pages 225–231, Sydney, Australia, August 1991.