Quantitative Modeling of Complex Computational Task Environments [*]

Keith Decker and Victor Lesser

May 10, 1993

Department of Computer Science
University of Massachusetts
Amherst, MA 01003
Email: DECKER@CS.UMASS.EDU
Phone: 413–545–3444
Fax: 413–545–1249

Technical Report 93-21

Abstract

There are many formal approaches to specifying how the mental state of an agent entails that it perform particular actions. These approaches put the agent at the center of analysis. For some questions and purposes, it is more realistic and convenient for the center of analysis to be the task environment, domain, or society of which agents will be a part. This paper presents such a task environment-oriented modeling framework that can work hand-in-hand with more agent-centered approaches. Our approach features careful attention to the quantitative computational interrelationships between tasks, to what information is available (and when) to update an agent's mental state, and to the general structure of the task environment rather than single-instance examples. A task environment model can be used for both analysis and simulation, it avoids the methodological problems of relying solely on single-instance examples, and provides concrete, meaningful characterizations with which to state general theories. This paper is organized around an example model of cooperative problem solving in a distributed sensor network.

[*]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.

1 Introduction

This paper presents a framework, TÆMS (Task Analysis, Environment Modeling, and Simulation), with which to model complex computational task environments that is compatible with both formal agent-centered approaches and experimental approaches. The framework allows us to both analyze and quantitatively simulate the behavior of single or multi-agent systems with respect to interesting characteristics of the computational task environments of which they are part. We believe that it provides the correct level of abstraction for meaningfully evaluating centralized, parallel, and distributed control algorithms, negotiation strategies, and organizational designs. No previous characterization formally captures the range of features, processes, and especially interrelationships that occur in computationally intensive task environments.

We use the term *computational task environment* to refer to a problem domain in which the primary resources to be scheduled or planned for are the computational processing resources of an agent or agents, as opposed to physical resources such as materials, machines, or men. Examples of such environments are distributed sensor networks, distributed design problems, complex distributed simulations, and the control processes for almost any distributed or parallel AI application. A job-shop scheduling application *is not* a computational task environment. However, the **control**[1] of multiple distributed or large-grain parallel processors that are jointly responsible for solving a job shop scheduling problem *is* a computational task environment. Distributed sensor networks use resources (such as sensors), but these resources are typically not the primary scheduling consideration. Computational task environments are the problem domain for control algorithms like many real-time and parallel local scheduling algorithms [1, 17, 23] and distributed coordination algorithms [9, 14].

The *reason* we have created the TÆMS framework is rooted in the desire to produce general theories in AI [5]. 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 careful specification of the 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 supports the graphical display of generated subjective and objective task structures, agent actions, and statistical data collection in CLOS on the TI Explorer.

The basic form of the computational task environment framework—the execution of interrelated computational tasks—is taken from several domain environment simulators [3, 4, 14]. If this were the only impetus, the result might have been a simulator like Tileworld [22]. However, formal research into multi-agent problem solving has been productive in specifying formal properties, and sometimes algorithms, for the control

[1]Planning and/or scheduling of computation.

process by which the mental state of agents (termed variously: beliefs, desires, goals, intentions, etc.) causes the agents to perform particular actions [6, 25, 26]. This research has helped to circumscribe the behaviors or actions that agents can produce based on their knowledge or beliefs. The final influence on TÆMS was the desire to avoid the individualistic agent-centered approaches that characterize most AI (which may be fine) and DAI (which may not be so fine). The concept of agency in TÆMS is based on simple notions of *execution*, *communication*, and *information gathering*. An agent is a locus of belief (state) and action. 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).

Section 2 will discuss the general nature of the three modeling framework layers. Sections 3 through 5 discuss the details of the three levels, and are organized around a model built with this framework for the study of organizational design and coordination strategies in a multi-agent distributed sensor network environment.

2 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 a 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 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 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.

The model of environmental and task characteristics proposed has three levels: *objective*, *subjective*, and *generative*. The *objective* level describes the essential, ‘real’ task structure of 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 [15], without the information about particular agents. The *subjective* level describes how agents view and interact with the 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 while individual behavior and system performance can be measured objectively, agents must make decisions with only subjective information.[2]

[2]In organizational theoretic terms, subjective *perception* can be used to predict agent actions or *outputs*,

Finally, the *generative* level describes the statistical characteristics required to generate the objective and subjective situations in a domain.

3 Objective Level

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 model is that task groups appear in the environment at some frequency, and induce tasks T to be executed by the agents under study. Task groups are independent of one another, but tasks within a single task group have interrelationships. Task groups or tasks may have deadlines $D(T)$. The *quality* of the execution or result of each task influences the *quality* of the task group result $Q(T)$ in a precise way (Section 3.1). 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 previous task executions ('non-local effects').[3] The basic purpose of the objective model is to formally specify how the execution and timing of tasks affect this measure of quality.

3.1 Local Effects: The *Subtask* Relationship

Task or task group quality ($Q(T)$) 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 $Q(T, t)$: [tasks $\times$ times] $\mapsto$ [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).

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 $\mathsf{subtask}(T_1, \mathbf{T}, Q_{\text{min}})$, then $Q(T_1, t) = Q_{\text{min}}(T_1, t) = \min_{T \in \mathbf{T}} Q(T, t)$. In this case the quality that is associated with task T_1 is the minimum quality associated with any of its subtasks. This is sometimes referred to as an AND because the quality of the parent remains at a minimum until every subtask has been completed. Other functions may be used for modeling particular environments.[4] Functions like sum and average indicate

but unperceived, objective environmental characteristics can still affect performance (or *outcomes*) [24].

[3]When local or non-local effects exist between tasks that are known by more than one agent, we call them *coordination relationships*[9]

[4]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

the possibility that not all tasks in the environment need to be carried out. We have now described how quality is modeled at tasks that have subtasks, and now turn our attention to methods.

3.2 Local Effects: Method Quality

Each method M at a time t will potentially produce (if executed by an agent, see Section 4.3) 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 3.3). 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).[5]

Let $\mathsf{P}(M,t)$ be the current amount of progress on the execution of M. If M were not interruptible and $\mathsf{S}(M)$ and $\mathsf{F}(M)$ were the execution start time and finish time, respectively, of M, then:

$$\mathsf{P}(M,t) = \begin{cases} 0 & t \leq \mathsf{S}(M) \\ t - \mathsf{S}(M) & \mathsf{S}(M) < t < \mathsf{F}(M) \\ \mathsf{F}(M) - \mathsf{S}(M) & t \geq \mathsf{F}(M) \end{cases}$$

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

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

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 [21]. 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 [1] and the design-to-time (approximate processing) camp[12, 17]. We can represent their ideas succinctly; in the anytime algorithm model partial results are always available,[6] as in the definition of $Q_{\text{lin}}(M,t)$ 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 $Q_{\text{DTT}}(M,t)$:[7]

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

(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 [2, 13].

[5]We model the effect of interruptions, if any, and the reuse of partial results as non-local effects (see Section 3.3).

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

[7]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.

3.3 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, and *quality effects* where $\mathbf{q}(M, t)$ (maximum quality) is changed. An effect class e is thus a function $e(T, M, t, d, q, p_1, p_2, \ldots)$: [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.[8] 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 NLE, then the effects are applied one after the other in an order specified in the model (the default is for effects with antecedents closer in the task structure to M to be applied first).

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:

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

[8] Other parameters, such as channel reliability, are being considered. The description of an agent's control and coordination algorithms will describe when and where communication actually occurs (see communication actions in Section 4.3, and the concept of agency in Section 4.1).

The current duration function **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 [16]. The clear specification of such concepts is one of the benefits of using our framework.

3.3.1 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. Typically a model will define different classes of effects, such as *causes*, *facilitates*, *cancels*, *constrains*, *inhibits*, and *enables* [10]. 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* [9, 10].

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 $\mathbf{q}(M,t) = \mathbf{q}_0(M)$.

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

Facilitates. Another effect, used by the PGP algorithm [15] but never formally defined, is the *facilitates* effect. Intuitively, one task may provide results to another task that *facilitate* the second task by decreasing the duration or increasing the quality of its partial result. Therefore the *facilitates* effect has two parameters (called *power* parameters) $0 \leq \phi_{\mathrm{d}} \leq 1$ and $0 \leq \phi_{\mathrm{q}} \leq 1$, 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).

$$\begin{aligned} R(T_a, s, t) &= \frac{Q_{\mathrm{avail}}(T_a, s)}{\mathbf{q}(T_a, t)} \\ \mathbf{facilitates}(T_a, M, t, d, q, \phi_{\mathrm{d}}, \phi_{\mathrm{q}}) &= \begin{cases} [d(1 - \phi_{\mathrm{d}} R(T_a, t, t)), & \\ q(1 + \phi_{\mathrm{q}} R(T_a, t, t))] & t < \mathsf{S}(M) \\ [d(1 - \phi_{\mathrm{d}} R(T_a, \mathsf{S}(M), t)), & \\ q(1 + \phi_{\mathrm{q}} R(T_a, \mathsf{S}(M), t))] & t \geq \mathsf{S}(M) \end{cases} \end{aligned} \tag{2}$$

So if T_a is completed with maximal quality, and the result is received before M is started, then the duration $\mathbf{d}(M,t)$ will be decreased by a percentage equal to the duration power ϕ_{d} of the *facilitates* effect. The second clause of the definition indicates that communication after the start of processing has no effect. In other work [9] we explored the effects on coordination of a *facilitates* effect with varying duration power ϕ_{d}, and with $\phi_{\mathrm{q}} = 0$.

Hinders. The *hinders* effect is the opposite of *facilitates*, because it increases the duration and decreases the maximum quality of the consequent. This can be used as a high-level model of distraction.

Precedence. We define the *precedence* effect as a combination of *enables* and *hinders*. 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 ($\phi_{\mathrm{d}} \geq 0$ and $\phi_{\mathrm{q}} \geq 0$). 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.

$$\textbf{precedes}(T_a, M, t, d, q, \phi_{\mathrm{d}}, \phi_{\mathrm{q}}) = \begin{cases} [\infty, 0] & [[Q_{\mathrm{avail}}(T_a, t) = 0] \wedge [t < \mathsf{S}(M)]] \\ & \vee [[Q_{\mathrm{avail}}(T_a, \mathsf{S}(M)) = 0] \wedge [t \geq \mathsf{S}(M)]] \\ [\mathbf{d}_0(M)/R(T_a, t, t)^{\phi_{\mathrm{d}}}, & \\ \mathbf{q}_0(M)R(T_a, t, t)^{\phi_{\mathrm{q}}}] & t < \mathsf{S}(M) \\ [\mathbf{d}_0(M)/R(T_a, \mathsf{S}(M), t)^{\phi_{\mathrm{d}}}, & \\ \mathbf{q}_0(M)R(T_a, \mathsf{S}(M), t)^{\phi_{\mathrm{q}}}] & t \geq \mathsf{S}(M) \end{cases} \tag{3}$$

3.4 Objective Modeling Example

Now that we have discussed the basic components of an objective model, let us turn to an example in which we 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 [14]. 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. This is the classic type of result that the TÆMS framework was created to address—we wish to *explain* this result, and better yet, to *predict* which algorithm will do the best in each situation. The level of detail to which you build your model will depend on the question you wish to answer—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, 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.

In our model of DSN problems, each vehicle track is modeled as a task group. 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 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 $t+1$; (1) vehicle track completion method (VCM) that represents merging all the VTMs together into a complete vehicle track hypothesis. Non-local enablement effects exist as shown in Figure 1—two VLMs *enable* each VTM, and all VTMs *enable* the lone VCM.

We have used this model to develop expressions for the expected value of, and confidence intervals on, the time of termination of a set of agents in any arbitrary DSN environment that has a static organizational structure and coordination algorithm [11]. 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. These results were achieved

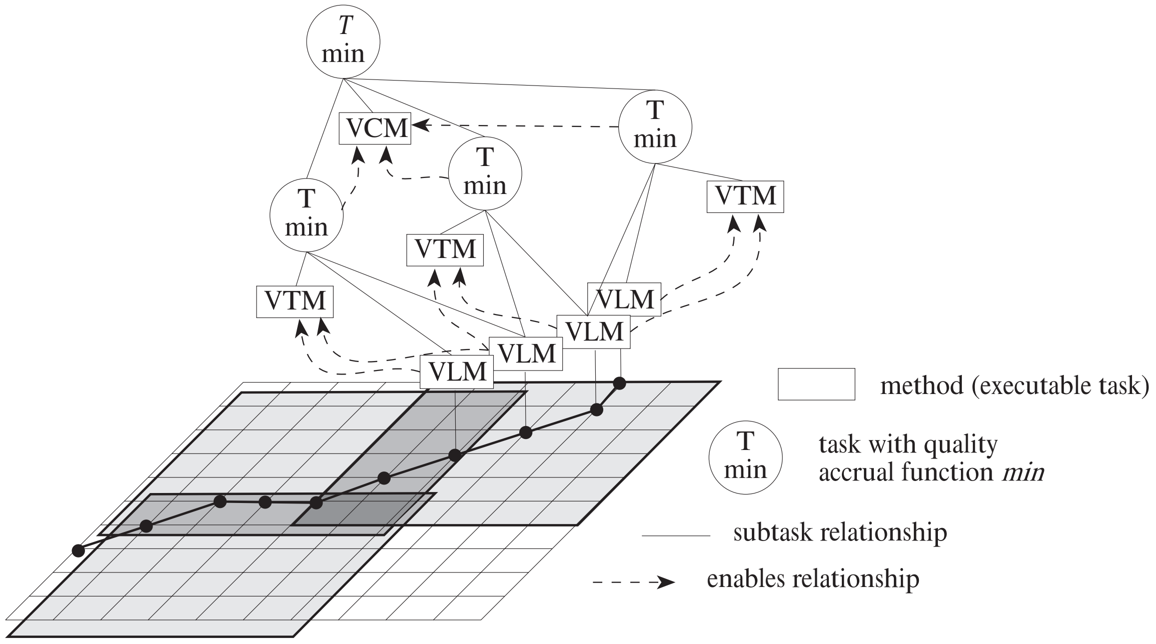

Figure 1: Objective task structure associated with a single vehicle track.

by direct mathematical analysis of the model and verified through simulation in TÆMS. We will give a summary of these results later in the paper (Section 5), after discussing the subjective and generative levels.

3.4.1 Expanding the Model

We will now add some complexity to the model. The length of a track l_i above is a *generative* level parameter. Given a set of these generative parameters, we can construct the objective model for a specific problem-solving instance, or *episode*. Figure 1 shows the general structure of episodes in our DSN environment model, rather than a particular episode. To display an actual objective model, let us assume a simple 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 [12, 8] for this and other approximate methods). 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 2 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 [14] is that redundant work is not generally useful; this is indicated by using *max* as the combination function for each agent's redundant methods. We could alter this assumption by simply changing this function (to *mean*, for example). 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. Sharing of results could be only one-way between methods.

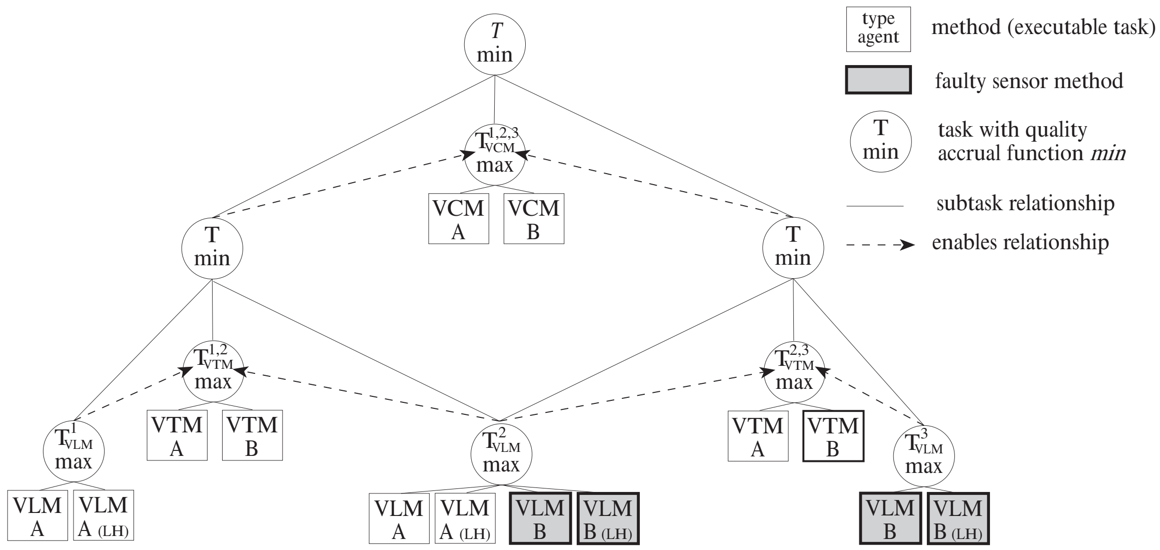

Figure 2: Objective task structure associated with two agents

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 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 *facilitation* 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 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.[9] 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. The more tracking methods are executed, the easier the remaining ones become. The implications of this model are that in a multi-agent

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

episode, then, the question becomes when to communicate partial results to another agent: the later an agent delays communication, the more the potential impact on the other agent, but the more the other agent must delay. We examined this question somewhat in [9].

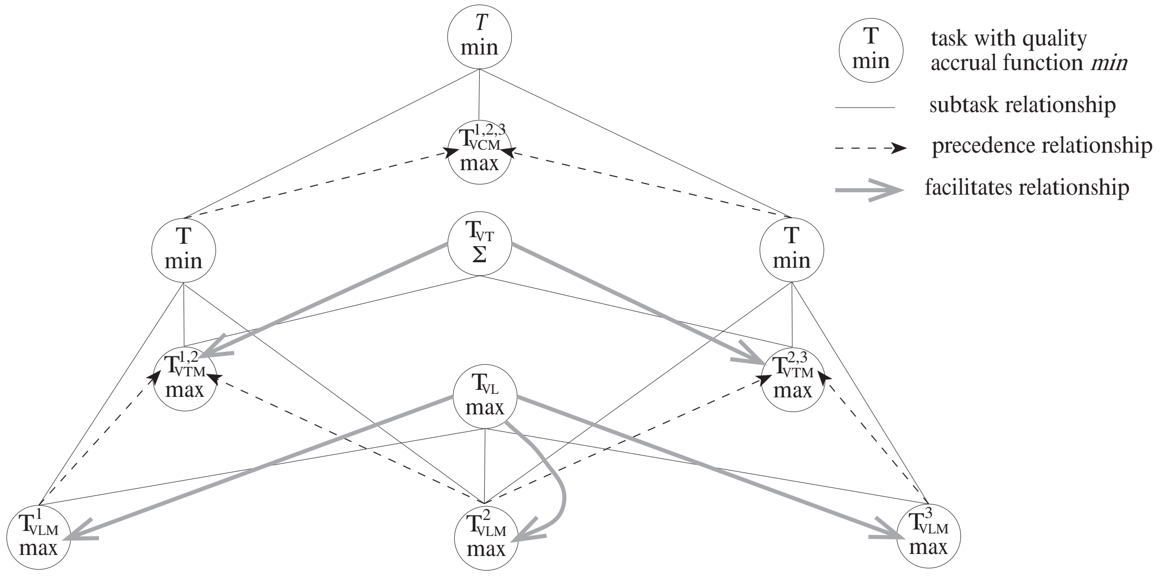

Figure 3: Non-local effects in the objective task structure

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.

4 Subjective Level

The purpose of a subjective level model of an environment is to describe what portions of the objective model of the situation 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.

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, but there are many formal and informal descriptions of the concept of agency (see [18, 19]). 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 an agent believes about its environment". Our view is that an "agent" is a locus of belief and action (such as computation).

The form of the rest of this section is as follows: how does the environment affect the beliefs of the agents; how do the beliefs of agents affect their actions, and how do the actions affect the environment.

4.1 Agent Beliefs

We use the symbol Γ_A^t to denote the set of beliefs of agent A at time t. A subjective mapping of an objective problem solving situation $\mathcal{O}$ is a function $\varphi : [A \times \mathcal{O}] \mapsto \Gamma_A$ from an agent and objective assertions to the beliefs of an agent. 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 **q** (maximum quality of a method), **d** (duration of a method), deadlines, and the relations subtask, nle, and comm.

4.2 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 (but see Section 5.2). The agent's control mechanism uses the agent's current set of beliefs Γ_A to update three special subsets of these beliefs (alternatively, *commitments*[25]) identified as the sets of information gathering, communication, and method execution actions to be computed.

The models we build typically further divide control into *local scheduling* and *coordination* (see [9]), 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.[10]

4.3 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: control, information gathering, communication, and method execution. First the control mechanisms assert (commit to) 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 [7], 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.

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

[10]Understanding the details of the control costs of particular algorithm implementations is important, but usually not at early stages of research. Detailed information about control costs can be used by TÆMS if it is available.

method execution, and execution with monitoring, suspension, and preemption. These follow from the discussion of Q_{DTT} and Q_{lin} in Section 3.2, 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 $\mathsf{S}(M)$ and remains in that state until the time t when $t - \mathsf{S}(M) = \mathbf{d}(M, t)$. Method execution actions are similar to what Shoham terms 'private actions' like DO[25].

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 [17].

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 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.

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 one 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 set of beliefs). A third kind of information gathering may identify coordination relationships—non local effects that span multiple agents. Both communication and information gathering actions take some period of time (not necessarily constant) to execute, as specified in the model.

4.4 Subjective Modeling Example

Let's return to the example we began in Section 3.4 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 $\varphi(A, \mathbf{d}_0(\text{faulty-VLM})) = \mathbf{d}_0(\text{VLM})$).[11] Other subjective level artifacts that are seen in [14] 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

[11] 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 replans accordingly.

initial data is sensed can be modeled by lowering the subjectively perceived value of the maximum quality **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).

Next we turn to the control of the agents in an environment. As an example, we will now present a very simple static control algorithm that uses no meta-level communication (more information about this and other algorithms can be found in [11]). In a static organization, agents divide their overlapping areas of responsibility as evenly as possible, resulting in new areas of responsibility for each agent with no overlap (to avoid redundant processing). Given a subjectively believed task structure as described in Section 3.4 and any communicated task results received by information gathering actions, the agent can at any time build a list of currently executable methods (under the set of precedence constraints). At any time an agent can also 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. Each agent follows the same control algorithm, 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):

```
(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))
```

It would be useful to know the performance of a system using this coordination algorithm. 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, 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 $(\mathbf{d}_0(C))$.

We have explained the objective and subjective levels of our modeling framework, and presented an example of a moderately complex task structure and a simple control algorithm for an agent that can accomplish this task. Next we turn to the generative level, where we specify the statistical properties of an environment across many episodes.

5 Generative Level

By using the objective and subjective levels of TÆMS we can model any *individual* situation; adding a *generative* level to the 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. Our previous work has created generative models of task interarrival times (exponential distribution), amount of work in a task cluster (Poisson), task durations (exponential), and the likelihood of a particular non-local effect between two tasks [11, 9, 17]. Generative level statistical parameters can also be used by agents in their subjective reasoning, for example, an agent may make control decisions based on the 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 [20]. Detailed model verification will be more important when using our framework to optimize parameters in a real application, as opposed to learning the general effects of parameters on a coordination or negotiation algorithm (see Section 6).

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 be 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 $\mathcal{D} =< A, \eta, r, o, \mathcal{T} >$ 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[12], and $\mathcal{T}$ specifies the homogeneous task group structure (as discussed in Sections 3.4 and 4.4). 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 a Poisson distribution with an expected value of η. If we were to extend this generative model to cover every feature we added to the objective and subjective models in Sections 3.4 and 4.4, we would need to add the likelihood of a sensor being faulty (noisy), the likelihood of a ghost track, etc.

5.1 Analysis Summary

We briefly showed at the end of Section 4.4 how we can predict the performance of a system given the objective and subjective models of a particular episode. This is very useful for explaining or predicting agent behavior in a particular episode or scenario, but not over many episodes in a real environment. To do this, we need to build probabilistic models of the relevant objective and subjective parameters (now viewed as random variables) that are based on generative level parameters. Our companion paper, [11], details this process, and

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

shows how the distributions of objective parameters such as "the number of VLM methods seen by the maximally loaded agent" ($\hat{S}$) and "the max number of task groups seen by the same agent" ($\hat{N}$) can be defined from just the generative parameters $\mathcal{D} =< A, \eta, r, o, \mathcal{T} >$.

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, as was discussed in Section 4.4:

$$\hat{S}\mathbf{d}_0(\text{VLM}) + (\hat{S} - \hat{N})\mathbf{d}_0(\text{VTM}) + (a-1)\hat{N}\mathbf{d}_0(\text{VTM}) + \hat{N}\mathbf{d}_0(\text{VCM}) + 2\mathbf{d}_0(I) + 2\mathbf{d}_0(C) \qquad (4)$$

We can use Eq. 4 as a predictor by combining it with the probabilities for the values of $\hat{S}$ and $\hat{N}$. Again, we refer the interested reader to [11] for derivations, verification, and applications of these results. Note that if the assumptions behind our generative model change (for instance, if we assume all agents initially line up side-by-side, instead of in a square, or if vehicles made a loop before exiting the sensed area) the probability distributions for $\hat{S}$ and $\hat{N}$ might change, but that the form of Eqn. 4 does not. If the agent's coordination algorithm changes, then Eqn. 4 will change (see [11]).

5.2 Simulation

Simulation is a useful tool for learning parameters to control algorithms, for quickly exploring the behavior space of a new control algorithm, and for conducting controlled, repeatable experiments when direct mathematical analysis is unwarranted or too complex. The simulation system we have built for the direct execution of models in the TÆMS framework supports, for example, the collection of paired response data, where different or ablated coordination or local scheduling algorithms can be compared on identical instances of a wide variety of situations (generated using the generative level of the model). We have used simulation to explore the effect of exploiting the presence of *facilitation* between tasks in a multi-agent real-time environment where no quality is accrued after a task's deadline [9]. The environmental generative characteristics here included the mean interarrival time for tasks, the likelihood of one task facilitating another, and the strength of the facilitation (ϕ_d).

The TÆMS framework is not limited to experimentation in distributed problem solving. In [17], Garvey and Lesser used the framework to describe the effects of various task environment and agent design features on the performance of their real-time 'design-to-time' algorithm. They manipulate the objective-to-subjective mapping to examine questions about monitoring the execution of tasks when true method durations are not known. They show that monitoring does provide a reduction in missed deadlines but that this reduction may be significant only during 'medium' loads. Garvey is now using a more complex model of *enabling* and *hindering* task structures to design an optimal design-to-time algorithm for certain task environments.

6 Conclusions

This paper has presented TÆMS, a framework for modeling computationally intensive task environments. TÆMS exists as both a language for stating general hypotheses or theories and as a system for simulation. The important features of TÆMS include its layered description of environments (*objective* reality, *subjective* mapping to agent beliefs, *generative* description of the other levels across single instances); its acceptance of any performance criteria (based on temporal location and *quality* of task executions); and its non-agent-centered point of view that can be used by researchers working in either formal systems of mental-state-induced behavior or experimental methodologies. TÆMS provides environmental and behavioral structures and features with which to state and test theories about the control of agents in complex computational domains, such as how decisions made in scheduling one task will affect the utility and performance characteristics of other tasks.

TÆMS is not only a mathematical framework, but also a simulation language for executing and experimenting with models directly. The TÆMS simulator supports the graphical display of generated subjective and objective task structures, agent actions, and statistical data collection in CLOS on the TI Explorer. These features help in both the model-building stage and the verification stage. The TÆMS simulator is being used not only for research into the coordination of distributed problem solvers[11, 9, 10], but also for research into real-time scheduling of a single agent[17], scheduling at an agent with parallel processing resources available, and soon, learning coordination algorithm parameters.

TÆMS does not at this time automatically learn models or automatically verify them. While we have taken initial steps at designing a methodology for verification (see [11]), this is still an open area of research [5]. Our future work will include building new models of different environments that may include physical resource constraints, such as airport resource scheduling. The existing framework may have to be extended somewhat to handle consumable resources. Other extensions we envision include specifying dynamic objective models that change structure as the result of agent actions. We also wish to expand our analyses beyond the questions of scheduling and coordination to questions about negotiation strategies, emergent agent/society behavior, and organizational self-design.

References

[1] Mark Boddy and Thomas Dean. Solving time-dependent planning problems. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, August 1989.

[2] Piero P. Bonissone and Keith S. Decker. Selecting uncertainty calculi and granularity: An experiment in trading-off precision and complexity. In L. N. Karnak and J. F. Lemmer, editors, *Uncertainty in Artificial Intelligence*. North Holland, 1986.

[3] Norman Carver and Victor Lesser. A new framework for sensor interpretation: Planning to resolve sources of uncertainty. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 724–731, August 1991.

[4] 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.

[5] 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.

[6] Philip R. Cohen and Hector J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42(3), 1990.

[7] 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), 1993.

[8] 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), 1993.

[9] Keith S. Decker and Victor R. Lesser. Analyzing a quantitative coordination relationship. COINS Technical Report 91–83, University of Massachusetts, November 1991. To appear in the journal *Group Decision and Negotiation*, 1993.

[10] Keith S. Decker and Victor R. Lesser. Generalizing the partial global planning algorithm. *International Journal of Intelligent and Cooperative Information Systems*, 1(2), June 1992.

[11] 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*, Chambéry, August 1993.

[12] 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.

[13] 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.

[14] Edmund H. Durfee, Victor R. Lesser, and Daniel D. Corkill. Coherent cooperation among communicating problem solvers. *IEEE Transactions on Computers*, 36(11):1275–1291, November 1987.

[15] E.H. Durfee and V.R. Lesser. Partial global planning: A coordination framework for distributed hypothesis formation. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(5):1167–1183, September 1991.

[16] 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*, Washington, July 1993.

[17] Alan Garvey and Victor Lesser. Design-to-time real-time scheduling. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(6), 1993. Special Issue on Scheduling, Planning, and Control.

[18] Les Gasser. Social conceptions of knowledge and action. *Artificial Intelligence*, 47(1):107–138, 1991.

[19] Carl Hewitt. Open information systems semantics for distributed artificial intelligence. *Artificial Intelligence*, 47(1):79–106, 1991.

[20] Jack P. C. Kleijnen. *Statistical Tools for Simulation Practitioners*. Marcel Dekker, New York, 1987.

[21] 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.

[22] 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, July 1990.

[23] 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.

[24] W. Richard Scott. *Organizations: Rational, Natural, and Open Systems*. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1987.

[25] 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.

[26] 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.