RL-TR-97-242
Final Technical Report
March 1998

DISTRIBUTED PLANNING AND CONTROL FOR APPLICATIONS IN TRANSPORTATION SCHEDULING

Brown University

Sponsored by
Advanced Research Projects Agency
ARPA Order No. 7684

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-97-242 has been reviewed and is approved for publication.

APPROVED:

WAYNE A. BOSCO
Project Engineer

FOR THE DIRECTOR:

WARREN H. DEBANY, JR., Technical Advisor
Command, Control, & Communications Directorate

If your address has changed or if you wish to be removed from the Air Force Research Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify AFRL/IFTB, 525 Brooks Road, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

ALTHOUGH THIS REPORT IS BEING PUBLISHED BY AFRL, THE RESEARCH WAS ACCOMPLISHED BY THE FORMER ROME LABORATORY AND, AS SUCH, APPROVAL SIGNATURES/TITLES REFLECT APPROPRIATE AUTHORITY FOR PUBLICATION AT THAT TIME.

DISTRIBUTED PLANNING AND CONTROL FOR APPLICATIONS IN TRANSPORTATION SCHEDULING

Thomas Dean

Contractor: Brown University
Contract Number: F30602-91-C-0041
Effective Date of Contract: June 1991
Contract Expiration Date: May 1995
Program Code Number: 4E20
Short Title of Work: Distributed Planning and Control for Applications in Transportation Scheduling

Period of Work Covered: Jun 91 - May 95

Principal Investigator: Thomas Dean
Phone: (401) 863-7645
AFRL Project Engineer: Louis J. Hoebel
Phone: (315) 330-3655

Approved for public release; distribution unlimited.

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by Louis J. Hoebel, AFRL/IFTB, 525 Brooks Road, Rome, NY.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1998	Final Jun 91 - May 95

4. TITLE AND SUBTITLE

DISTRIBUTED PLANNING AND CONTROL FOR APPLICATIONS IN TRANSPORTATION SCHEDULING

5. FUNDING NUMBERS

C - F30602-91-C-0041
PE - 62702F
PR - G684
TA - 00
WU - 04

6. AUTHOR(S)

Thomas Dean

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Brown University
Computer Science Department
P.O. Box 1910
Providence, RI 02912

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington, VA 22203-1714

AFRL/IFTB
525 Brooks Road
Rome, NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

RL-TR-97-242

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Wayne Bosco/IFTB/(315) 330-4833

12a. DISTRIBUTION AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The research addresses combinatorial problems in transportation planning and scheduling. This work addresses fundamental problems n real-time planning and crisis decision making. Problems of uncertainty and incomplete information are dealt with by employing a stochastic domain model called a Markov decision process (MDP). Complexity in decision making is dealt with by using iterative techniques and decision theory to allocate computational resources at runtime. MASE is a simulation and development environment for problems involving multiple interacting agents. MASE provides robust and efficient communication and negotiation facilities to support distributed solutions to planning and utilities problems. MASE or its underlying communication subsystem running in a stand-alone configuration can be used by distributed components implemented in C, C++, or Common Lisp. Discusses a complementary approach to envelope-based methods for solving large MDPs, i.e., MDPs involving a very large number of states. This approach works by decomposing a large MDP into several smaller MDPs which are weakly coupled. A solution is obtained by solving each of the smaller MDPs individually and then combining these local solutions to obtain a global solution. An overview is provided on a proposed new direction of research for embedded planning and scheduling. Described is a general model for embedded planning and control systems. This model describes the rudiments of a software specification framework that is critical to any significant progress on real-time systems of any complexity.

14. SUBJECT TERMS

Scheduling System, Distributive Planning, Markov decision process (MDP), Transportation Scheduling, Crisis Planning, MASE

15. NUMBER OF PAGES

56

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

List of Figures

1 Introduction and Overview

This report describes research conducted at Brown University during the period from June 1991 to June 1995. Originally, the contract was due to expire on May 31, 1994, but we applied for and were granted a one-year, no-cost extension. The research was supported by the Air Force and the Advanced Research Projects Agency of the Department of Defense under contract number F30602-91-C-0041. The research addresses combinatorial problems in transportation planning and scheduling. The results of this research are summarized in the four self-contained sections that make up the bulk of this report.

In Section 2, we describe our research on solving combinatorial planning problems under time constraints. This work addresses fundamental problems in real-time planning and crisis decision making. We deal with problems of uncertainty and incomplete information by employing a stochastic domain model called a *Markov decision process* (MDP). We deal with complexity in decision making by using iterative techniques and decision theory to allocate computational resources at runtime. In a concession to computational complexity and real-time constraints, we employ approximations rather than exact solutions and trade solution accuracy and optimality for time to make the best use of the time available for decision making. The various technical details behind this work appear in several papers, but the most comprehensive presentation is in *Planning Under Time Constraints in Stochastic Domains* (Thomas Dean, Leslie Kaelbling, Jak Kirman, and Ann Nicholson), Artificial Intelligence **76** (1995).

In Section 3. we describe MASE, a simulation and development environment for problems involving multiple interacting agents. MASE provides robust and efficient communication and negotiation facilities to support distributed solutions to planning and utilities problems. MASE or its underlying communication subsystem running in a stand-alone configuration can be used by distributed components implemented in C, C++, or Common Lisp. MASE is available for use within the planning initiative and both a programmers manual and users manual have been produced as part of our effort for this contract. The technical details behind this work appear in several articles and technical reports but the most up-to-date and comprehensive presentation will appear in *A Framework for the Development of Multi-Agent Architectures* (Moises Lejter and Thomas Dean), IEEE Expert.

In Section 4, we describe a complementary approach to our envelope-based methods for solving large MDPs, *i.e.,* MDPs involving a very large number of states. This approach works by decomposing a large MDP into several smaller MDPs which are weakly coupled. A solution is obtained by solving each of the smaller MDPs individually and then combining these *local* solutions to obtain a global solution. Technical details will appear in *Localized Temporal Reasoning Using Subgoals and Abstract Events* (Shieu-Hong Lin and Thomas Dean), Journal of Computational Intelligence.

In Section 5, we provide an overview of a new direction for our research on embedded planning and scheduling. We describe a general model for embedded planning and control systems. Using this model, we describe the rudiments of a software specification framework that we believe critical to any significant progress on real-time systems of any complexity. Note that most military applications and many real-world commercial applications have significant real-time components; the world will not wait for us while we try to compute exact solutions to the problems we are faced with.

Three graduate students earned Ph.D.s under this contract; they are Jak Kirman, Lloyd Greenwald, and Shieu-Hong Lin. Several undergraduates were involved in the research at one time or another; of these students, four went on to graduate school including one to MIT, two to Pennsylvania University, and one to the University of Washington. Ann Nicholson, who was a postdoctoral research scientist under this contract, is now an assistant professor at Monash University in Victoria, Australia.

2 Crisis Planning in Stochastic Domains

In a completely deterministic world, it is possible for a planner simply to generate a sequence of actions, knowing that if they are executed in the proper order, the goal will necessarily result. In nondeterministic worlds, planners must address the question of what to do when things do not go as expected. In tradition planning, the world is assumed to be deterministic for the purpose of planning, but its nondeterminism is accounted for by performing execution monitoring or by generating reactions for world states not on the nominal planned trajectory.

In our work, we address the problem of planning in nondeterministic domains by taking nondeterminism into account from the very start. There is already a well-explored body of theory and algorithms addressing the question of finding optimal policies (universal plans), which specify the best action to take for every possible situation which may arise during plan execution, for nondeterministic domains. Unfortunately, these methods are impractical in large state spaces. However, if we know the start state, and have a model of the nature of the world's nondeterminism, we can restrict the planner's attention to a set of world states that are likely to be encountered on the way to the goal. Furthermore, the planner can generate more or less complete plans depending on the time it has available. In this way, we provide efficient methods, based on existing techniques of finding optimal strategies, for planning under time constraints in non-deterministic domains. Our approach addresses the uncertainty resulting from control error, but not sensor error; in most of the following, we assume certainty in observations, but discuss relaxing this assumption in [10].

We assume that the environment can be modeled as a stochastic automaton: a set of states, a set of actions, and a matrix of transition probabilities. In our approach, constructing a plan to achieve a goal corresponds to finding a *policy* (a mapping from states to actions) that maximizes expected performance. Performance is based on the expected accumulated reward over sequences of state transitions determined by the underlying stochastic automaton. The rewards are determined by a *reward function* (a mapping from states to the real numbers) specially formulated for a given goal. A good policy in our framework corresponds to a universal plan for achieving goals quickly on average.

There are dynamic programming algorithms for computing the optimal policy given a stochastic model of the world. They are useful in small to medium-sized state-spaces, but become intractable on very large state-spaces. We address this difficulty by making some informal assumptions about the environments in which we are working that allow us to generate approximate solutions efficiently. In particular, we assume that the environment has the following properties:

- *high solution density*: it is relatively easy to find plausible (though perhaps not optimal) solutions
- *low dispersion rate*: from any given state, there are only a few states to which transitions can be made
- *continuity*: it is reasonable to estimate the values of states by considering the values of near-by states (where distance is measured as the expected number of steps between states)

Many large, realistic planning problems, such as those involving high-level navigation or scheduling, have these properties.

Our approach is motivated by the intuitively appealing work of Drummond and Bresina on 'anytime synthetic projection' [13]. We reformulate their basic framework in terms of Markov decision processes, cast the algorithmic issues in terms of approximations to specific optimization problems, provide a disciplined approach to allocating computational resources at run time, introduce techniques for specifying goals in stochastic domains, and describe how to extend the framework to deal with uncertainty in observation.

2.1 Markov Decision Models and Coping with Large State Spaces

Following the work on Markov decision processes [2, 4], we model the entire environment as a stochastic automaton, which we refer to as the *system automaton*. Let S be the finite set of world states; we assume that they can be reliably identified by the system. Let $\mathcal{A}$ be the finite set of actions; every action can be taken in every state. The transition model of the environment is a function mapping elements of $\mathcal{S} \times \mathcal{A}$ into discrete probability distributions over $\mathcal{S}$. We write $\Pr(s_1, a, s_2)$ for the probability that the world will make a transition from state s_1 to state s_2 when action a is taken.

A *policy* π is a mapping from $\mathcal{S}$ to $\mathcal{A}$, specifying an action to be taken in each situation. An environment combined with a policy for choosing actions in that environment yields a Markov chain [20].

A *reward function* is a mapping from $\mathcal{S}$ to $\Re$, specifying the instantaneous reward that the system derives from being in each state. Given a policy π and a reward function R, the *value* of state $s \in \mathcal{S}$, $V_\pi(s)$, is the sum of the expected values of the rewards to be received at each future time step, discounted by how far into the future they occur. The *discounting factor*, $0 \leq \gamma < 1$, controls the influence of rewards in the distant future.

One of the most common goals is to achieve a certain condition p *as soon as possible.* If we define the reward function as $R(s) = 0$ if p holds in state s and $R(s) = -1$ otherwise, and represent all goal states as being absorbing, then the optimal policy will result in the system reaching a state satisfying p as soon as possible. A state is *absorbing* if all actions result in that same state with probability 1; that is, $\forall a \in \mathcal{A},\ \Pr(s, a, s) = 1$. The language of reward functions is quite rich, allowing us to specify much more complex goals, including the maintenance of properties of the world and prioritized combinations of primitive goals; this is explored in [10].

Given a state-transition model, a reward function, and a value for γ, it is possible to compute the optimal policy using either the policy iteration algorithm [18] or the value iteration algorithm [2]. We use the policy iteration algorithm because it is guaranteed to converge in a finite number of steps — polynomial in $|\mathcal{S}|$, but generally a small number of steps in the domains that we have experimented with — and thus simplifies debugging our computational experiments.

As the size of our state spaces grows, even a polynomial-time algorithm such as policy iteration becomes too inefficient. We assume that our environment is such that, for any given reward function and initial starting state, it is sufficient to consider a highly-restricted subset of the entire state space in our planning.

A *partial policy* is a mapping from a subset of $\mathcal{S}$ into actions; the domain of a partial policy π is called its *envelope*, $\mathcal{E}_\pi$. The *fringe* of a partial policy, $\mathcal{F}_\pi$, is the set of states that are not in the envelope of the policy, but that may be reached in one step of policy execution from some state in the envelope. To construct a *restricted* stochastic automaton, we take an envelope $\mathcal{E}$ of states and add the distinguished state OUT, which is absorbing.

The cost of falling out of the envelope is a parameter that depends on the domain. For example, if it is possible to re-invoke the planner when the system falls out of the envelope, then one approach is to assign $V(\text{OUT})$ to be the estimated value of the state into which the system fell minus some function of the time required to construct a new partial policy.

2.2 Algorithms

The basic algorithm starts with an initial policy and a restricted state space (or *envelope*), extends that envelope, and then computes a new policy. We would like it to be the case that the new policy π' is an improvement over (or at the very least no worse than) the old policy π in the sense that $V_{\pi'}(s_0) - V_{\pi}(s_0) \geq 0$. In general, however, we cannot guarantee that the policy will improve without extending the state space to be the entire space of the system automaton, which results in computational problems. The best that we can hope for is that the algorithm improves in *expectation*. Although it is possible to construct system automata for which even this improvement in expectation is impossible, we believe many moderately benign environments are well-behaved in this respect. In particular, *navigation* environments (excluding mazes) and scheduling domains, in which transitions are restricted by spatio-temporal constraints, generally satisfy our requirements.

There are two basic types of operations on the restricted automaton. The first is called *envelope alteration* and serves to increase or decrease the number of states in the restricted automaton. The second is called *policy generation* and determines a policy for the system automaton using the restricted automaton. Note that, while the policy is constructed using the restricted automaton, it is a complete policy and applies to all of the states in the system automaton. For states outside of the envelope, the policy is defined by a set of *reflexes* that implement some default behavior for the system.

Precursor Deliberation Model In the precursor deliberation model, there are two separate phases of operation: planning and execution. The planner constructs a policy that is followed by the execution system until a new goal must be pursued or until the system falls out of the current envelope. In the simplest precursor models, a deadline is specified indicating when planning stops and execution begins. The high-level planning algorithm, given a description of the environment and start state s_0 or a distribution over start states, is as follows:

1. Generate an initial envelope $\mathcal{E}$
2. While ($\mathcal{E} \neq \mathcal{S}$) and (not deadline) do

a. Extend the envelope $\mathcal{E}$

b. Generate an optimal policy π for restricted automaton with state set $\mathcal{E} \cup \{\text{OUT}\}$

3. Return π

The algorithm first finds a small subset of world states and calculates an optimal policy over those states. Then it gradually adds new states in order to make the policy robust by decreasing the chance of falling out of the envelope. After new states are added, the optimal policy over the new envelope is calculated. Note the interdependence of these steps: the choice of which states to add during envelope extension may depend on the current policy, and the policy generated as a result of optimization may be quite different depending on which states were added to the envelope. The *multiple-rounds* of envelope alteration and policy generation are terminated when a deadline has been reached or when the envelope has been expanded to include the entire state space.

Recurrent Deliberation Model A more sophisticated model of interaction between planning and execution is one in which the planner runs concurrently with the execution, sending new or expanded strategies to the executor as they are developed. In recurrent-deliberation models, the system has to repeatedly decide how to allocate time to deliberation, taking into account new information obtained during execution. The details of such models are discussed in [8]; here we provide just a rough sketch. We assume two separate modules: one for planning and a second for execution. In the simplest model, the planner and executor operate in a rigid cycle with a period determined by fixed length of time. At the beginning of each cycle, the planner is given the current state by the execution module; the planner spends the fixed length of time working on a new policy; at the end of the fixed time, the planner gives the new policy to the execution module.

In the recurrent models, it is often necessary to remove states from the envelope in order to lower the computational costs of generating policies from the restricted automata. For instance, in the transportation scheduling domain, it may be appropriate to remove states corresponding to portions of a schedule the transportation vehicles have already executed, if there is little chance of returning to those states. Figure 1 shows a typical sequence of changes to the envelope corresponding to the state space for the restricted automaton. The current state is indicated by ◆ and the goal state is indicated by □.

The recurrent planning algorithm, given a description of the environment, the policy π_c that is currently being followed by the execution system, and the state of the system at the beginning of the planning interval, s_c, is as follows:

While (not goal) do

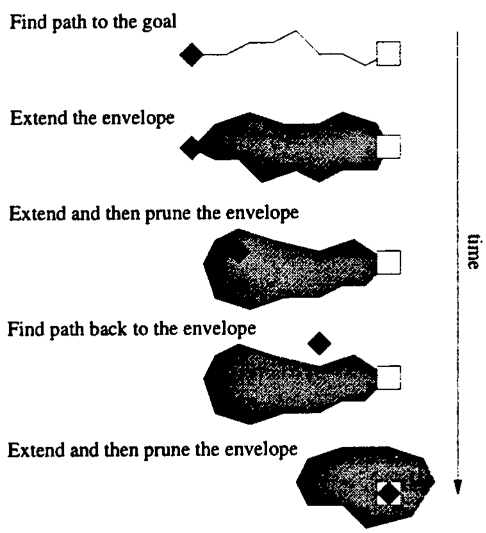

Figure 1: Typical sequence of changes to the envelope

1. Set s_c to be the current state for planning purposes
2. While (not end of current planning interval) do
   a. Extend the envelope $\mathcal{E}$
   b. Prune the envelope $\mathcal{E}$
   c. Generate an optimal policy π' for restricted automaton with state set $\mathcal{E} \cup \{\text{OUT}\}$
3. Set π_c to be the new policy π'

The details of the extension and pruning of the envelope will depend on the system's expected state at the end of the planning interval.

The subcomponents of the precursor and recurrent algorithms – initial trajectory planning, policy generation and envelope alteration – are described in [9]. Each subcomponent can be implemented as an anytime algorithm [7], one that can be interrupted at any point during execution to return an answer whose value, at least in certain classes of stochastic processes, improves in expectation as a function of the computation time. We cast the problem of allocating computational resources to the subcomponents as an optimization problem and use describe decision-theoretic techniques to compute approximations.

2.3 Deliberation Scheduling

Deliberation scheduling is the problem of allocating processor time to envelope alteration and policy generation. It is natural to think of deliberation scheduling in terms of optimization even if the combinatorics dictate that an optimal solution is not computationally feasible. Having said this, it still remains to determine what optimization problem we are trying to solve. We have to specify exactly what options are allowed and what information is available; such a characterization is generally referred to as a *decision model.*

In [8] we present a number of decision models. The algorithms given in Section 2.2 are examples of particular precursor deliberation and recurrent deliberation decision models. It should be pointed out that for each instance of the problems that we consider there are a large number of possible decision models. By specifying different decision models, we can make deliberation scheduling easy or hard. Our selection of which decision models to investigate is guided by our interest in providing insight into the problems of time-critical decision making and our anticipation of the combinatorial problems involved in deliberation scheduling. At present, we ignore the time spent in deliberation scheduling; for practical reasons, however, we are interested in decision models for which the on-line time spent in deliberation scheduling is negligible.

In recurrent deliberation models, the system has to decide repeatedly how to allocate time to deliberation, taking into account new information obtained during execution. In [8] we consider models for recurrent deliberation in which the system allocates time to deliberation only at prescribed intervals, which we call *discrete, weakly-coupled, recurrent deliberation* models. *Discrete* because each tick of the clock corresponds to exactly one state transition; *recurrent* because the execution module gets a new policy from the planning module periodically; *weakly coupled* in that the two modules communicate by having the execution module send the planning module the current state and the planning module send the execution module the latest policy. The intervals between planner-executor communication may be *fixed* or *variable* length.

In general, there are many more possible strategies for deploying envelope alteration and policy generation in recurrent models than in the case of precursor models. To cope with the attendant combinatorics, we raise the level of abstraction slightly and assume that we are given a small set of *deliberation strategies* that have been determined empirically to improve policies significantly in various circumstances. Each deliberation strategy corresponds to some fixed schedule for allocating processor time to envelope alteration and policy generation routines. For example, a strategy might consist of `findfirstpath` (find a first path to be the initial envelope), `robustify[20]` (add to the envelope the 20 fringe states most likely to be reached using the current policy), `optimize` (perform policy iteration until the optimal policy for the restricted automaton is generated), `prune[15]` (of the states that have a worse value than the current state, remove the 15 least likely to be reached using the current policy), `optimize`. Also, in anticipation of combinatorial issues that arise in our experimental studies, we adopt a simpler myopic decision model; we assume that the system will apply exactly one deliberation strategy and commit to the resulting policy thereafter.

2.4 Experimental Results

Greedy Precursor Deliberation In general, computing the optimal deliberation schedule for multiple-round precursor-deliberation models, such as that used in the algorithm in Section 2.2, computationally complex. We have experimented with a number of simple, greedy and myopic scheduling strategies; we report on one such strategy here.

We gathered a variety of statistics on how extending the envelope increases value. At run time, we use the size of the automaton and the estimated value of the current policy to index into a table of *performance profiles* giving expected improvement as a function of number of states added to the envelope. Using an experimental domain with 664 states, we generated 1,600,000 data points to compute these statistics plus estimates of the time required for one round of envelope alteration followed by policy generation given the size of the envelope, the number of states added, and value of the current policy. We use the following simple greedy strategy for choosing the number of states to add to the envelope on each round. For each round of envelope alteration followed by policy generation, we use the statistics to determine the number of states which, added to the envelope, maximizes the ratio of performance improvement to the time required for computation.

We compared the performance of (1) our planning algorithm using the greedy deliberation strategy with (2) policy iteration optimizing the policy for the whole domain. Our results show that the planning algorithm using the greedy deliberation strategy supplies a good policy early, and typically converges to a policy that is close to optimal before the whole domain policy iteration method does. Figure 2 shows average results from 620 runs, where a single run involves a particular start state and goal state. The graph shows the average value of the start state under the policy available at time t, $V_{\hat{\pi}}(s_0)$, as a function of time. In order to compare results from different start/goal runs, we show the average ratio of the value of the current policy to the value of the optimal policy for the whole domain, plotted against the ratio of actual time to the time, T_{opt}, that the policy iteration takes to reach that optimal value.

The greedy deliberation strategy performs significantly better than the standard optimization method. We also considered very simple strategies such as adding a small fixed number of fringe states each iteration, and adding the whole fringe each iteration, which

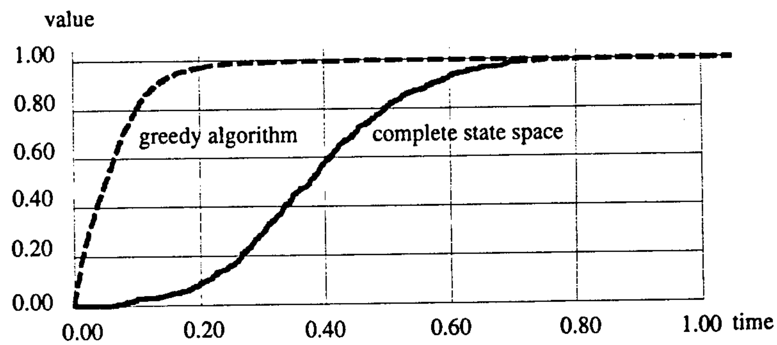

Figure 2: Comparison of planning algorithm using greedy deliberation strategy (dashed line) with the policy iteration optimization method (solid line): Average over 630 runs

performed fairly well for this domain, but not as well as the greedy policy. Further experimentation is required to draw definitive conclusions about the comparative performance of these deliberation strategies for particular domains.

Recurrent Deliberation We present results for recurrent-deliberation problems of indefinite duration using statistical estimates of the value of a variety of deliberation strategies. We use a discrete, weakly-coupled decision model with variable-length intervals for deliberation.

We gathered 600,000 data points for the same experimental domain, with state space size ranging from 632 to 15800 states, for 12 hand-crafted deliberation strategies. The start/goal pairs were chosen uniformly at random. We simulated execution in parallel with the planner until the goal was reached. The planner performed `findfirstpath (FFP)` to obtain the initial envelope, then entered the following loop: choose one of the 12 strategies uniformly at random, execute that strategy, and then pass the new policy to the executor.

We found the following conditioning variables to be significant: the envelope size, $|\mathcal{E}|$, the estimated value of the current state $\hat{V}_\pi$, the "fatness" of the envelope (the ratio of envelope size to fringe size), and the Manhattan distance, M, between the start and goal locations. We then built the lookup tables of the expected improvement in value as a function of the attributes $|\mathcal{E}|$, $\hat{V}_\pi$, the fatness, M and the strategy s. The lookup table granularity used was 3 buckets per attribute dimension.

To test our algorithm, we took 50 pairs of start and goal states, chosen uniformly at random. For each pair we ran the executor in parallel with the following deliberation mechanisms: recurrent-deliberation with strategies chosen using statistical estimates of

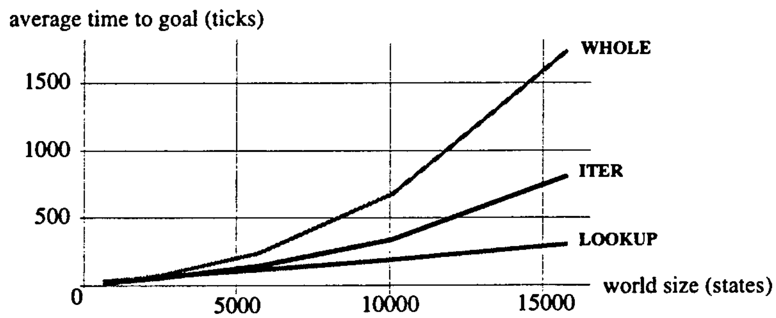

Figure 3: Comparison of the recurrent algorithm to policy iteration over varying domain size

their expected improvement in value (LOOKUP); dynamic programming policy iteration over the entire domain, with a new policy given to the executor, after each iteration (ITER), and only after it has been completely optimized (WHOLE).

Figure 3 shows the average number of steps taken by the system to reach the goal for the various algorithms. For the smaller domains, the recurrent-deliberation algorithm does not perform better than either of the policy iteration algorithms. However, as we move to larger domains, the improvement is marked. As we might expect, WHOLE is exponential and becomes computationally infeasible as the size of the domain increases. ITER also shows a non-linear degradation in the time to goal. LOOKUP shows linear behavior, clearly performing better as the domain size increases.

2.5 Further work

The implementation used to obtain the experimental results did not include a separate trajectory planning phase that looks for new paths to goal states. The addition of trajectory planning would reduce the extent to which the performance depended on the quality of the first path used as the initial envelope, and hence should improve the performance of the system as a whole.

In [10] we outline an extension of our approach to handle uncertainty in observation, based on the theory of partially observable Markov processes. We will be incorporating this into our empirical investigations, together with the more complex goals which can also be represented by reward functions.

We have described a general approach to planning in stochastic domains. We mentioned at the start of this section that we were making informal assumptions about the characteristics of the domains to which the general approach may apply — high solution density, low dispersion rate, and continuity. We are now developing a precise specification off domain characteristics that affect performance. This specification describes a space of problems defined by the domain characteristics. An analysis of correlations and dependencies among domain characteristics and how they affect performance will identify regions of this space to which our approach applies. We are developing an experimental environment that supports empirical verification of theoretical claims and assesses the applicability or our approach for specified domains.

3 Distributed Planning and Scheduling

We have been working on the development of MASE, an environment for the simulation of multiple interacting agents, as a vehicle for implementing software to test distributed algorithms for applications such as transportation scheduling and crisis management.

The principal goal behind MASE was to provide developers with a set of very powerful abstraction to facilitate the construction of distributed systems built using the agent model. As such, it follows previous work done by Gasser *et al* [15] and Green [16]. For a survey of distributed object-oriented systems in general, see [6].

MASE provides the following facilities to application developers:

- Message Exchange

  MASE provides developers with a powerful message exchange subsystem that agents can use to communicate with each other. The details of the actual implementation of the communications channel (such as the protocol to be used, the mechanisms to use in order to find the destination of any one message, or any details of local versus remote message delivery) are hidden from the application developers.

  The message exchange subsystem provides a useful set of abstractions for inter-process communications in its own right, which we call the CoRaCLe library.

- Conflict Resolution

  One of the first problems that must be addressed by developers of distributed control systems is this issue of conflict resolution, which involves the coordination of possibly conflicting goals generated by several of the agents in the system. MASE provides a default general-purpose conflict resolution strategy that developers may rely upon, using negotiation protocols based on game-theory.

- Temporal Reasoning

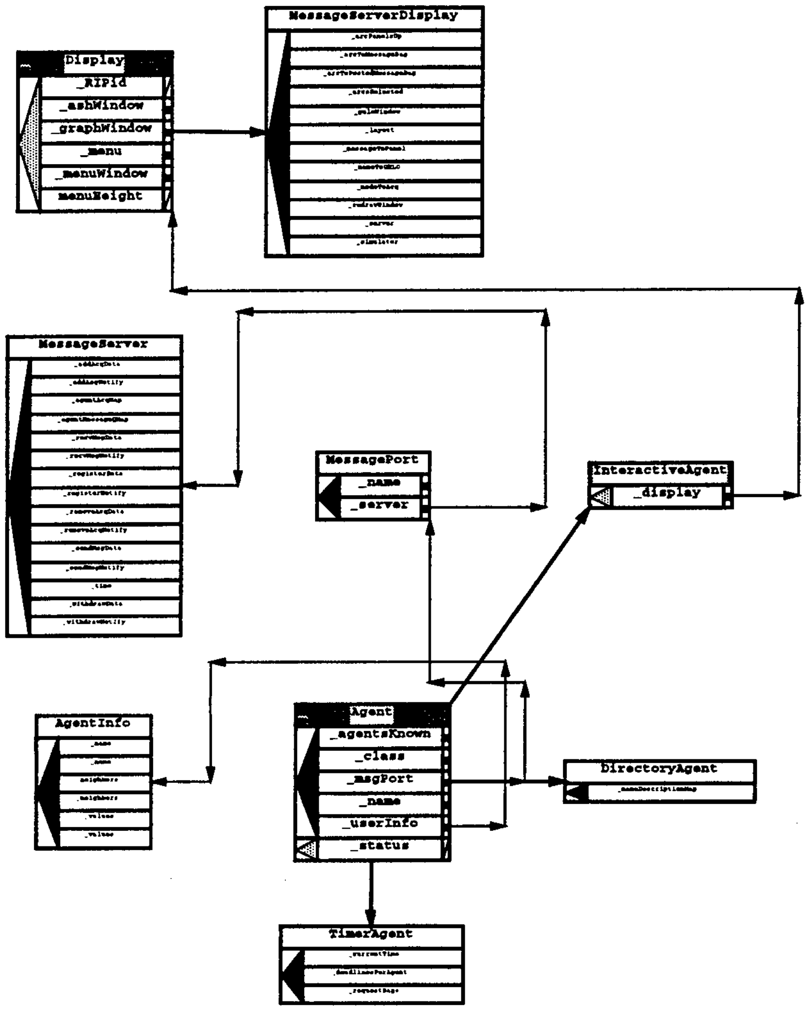

Figure 4: Class Agent Hierarchy and Helper Classes

MASE provides a simple general-purpose temporal reasoning engine that application developers may rely upon. This facility provides agents with the ability to store time-stamped information, and it automatically maintains the validity of the information stored as new information is added to the database.

- Development Support

  MASE provides additional facilities to ease the development process, such as an interactive, graphical interface to the simulation environment. This interface supports the interactive addition and deletion agents into a simulation run, single step and continuous simulation, interactive display and querying of the states of the agents in the system and the messages exchanged between them, and additional control over the characteristics of the communications channels that link the different objects (such as propagation delays over the different communications channels).

MASE supports several execution models:

- single-process, single-thread execution: all the agents defined in a simulation run in this model share the same process address space under UNIX. The simulator system executes the different agents cyclically, in a user-defined sequence.
- multiple-process, multiple-thread: each agent in the simulation runs in its own process under UNIX.

MASE is implemented as a library of classes written in C++ (including the communications facilities provided by CoRaCLe); it relies on the BWE windowing library [22] developed at Brown University for the graphical facilities of its user interface.

3.1 CoRaCLe: The Message Exchange Subsystem

The CoRaCLe message exchange subsystem is the component of MASE responsible for providing support for the exchange and automatic handling of messages by the agents. It provides abstractions to implement the different message-passing protocols supported by MASE in an implementation-independent fashion, freeing applications developers from concerns stemming from either the actual details of the message passing protocols supported by the underlying operating system or the particular execution environment chosen for a particular implementation.

CoRaCLe is an attempt to provide a basic set of powerful domain-independent communications abstractions for distributed processing. Its design goals are:

- to provide the fundamental communications abstractions necessary to develop distributed applications;
- to isolate developers as much as possible from the details of the underlying communications protocols at the operating systems or hardware levels;
- to provide a communications paradigm flexible enough to support dynamic distributed applications;

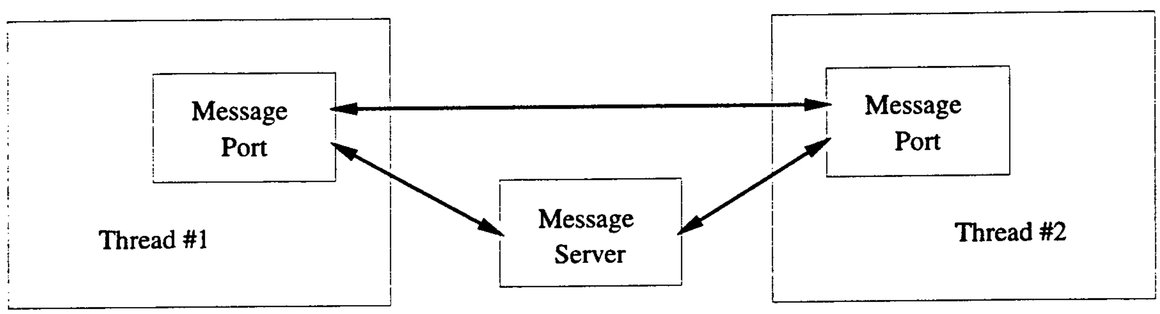

Figure 5: The Organization of CoRaCLe

- to be extensible enough to allow developers to build more complex, or application-specific, communications paradigms on top of the basic abstractions provided.

The CoRaCLe library supports a particular style of communications — one-way asynchronous message delivery — across a variety of underlying communications protocols. The library provides developers with a general-purpose communications module built around the notions of "named ports" that can exchange (send or receive) messages from any other CoRaCLe ports. CoRaCLe supports two modes of message exchange:

- Requests to send a message can be addressed to a specific other CoRaCLe port, by name;
- Ports can request to "eavesdrop" on the message traffic carried over CoRaCLe links.

The communications model CoRaCLe presents its clients postulates the existence of a central message exchange. This central exchange determines which message ports to use to forward messages to their destination ports and enables clients to "eavesdrop" on any of the message traffic that may be of interest to them, as shown in Figure 5. In fact, there need not be a physical central message repository. CoRaCLe allows its clients to choose among several underlying communications protocols transparently. Some protocols are in fact built around such a central repository (like the `Central` and `Local` versions), while others provide direct port-to-port communications (like the `Distributed` version) and simply provide a "virtual" message center, to preserve the semantics of the CoRaCLe communications model.

The message exchange subsystem supports two different kinds of message exchange protocols at the application developer's level:

- direct point-to-point communications (both one-to-one and one-to-many broadcasting) (implemented using either TCP/IP or RPC);

- communications via a shared tuple space in which messages are maintained. This mechanism is the one used in Linda [3], as well as the basis for blackboard systems [17].

These two protocols are enough to implement all commonly used communications protocols for distributed systems (see [1]). Although the other protocols can be implemented in terms of the two protocols listed above, MASE does not provide any other communications protocols itself.

3.2 Agent Abstractions

All facilities MASE provides to application developers are captured by the Agent abstraction provided by MASE. Application-specific agents are created as classes derived from this core Agent, thus inheriting all the facilities that abstraction provides.

The basic facilities provided by the MASE Agent abstraction, in addition to the support for inter-agent communications mentioned above, include automatic support for conflict resolution and a general-purpose temporal reasoning engine.

3.2.1 MASE support for Conflict Resolution

MASE provides a general-purpose algorithm for conflict resolution among multiple competing agents, based on some previous work on inter-agent negotiation [23], [24], [25], [21]. The approach allows the interacting agents to maximize their cooperation (ie, to maximize the number of actions of common interest) subject to the constraints imposed by the limited horizon resulting from constraints on the processing time available to the agents.

The default strategy MASE provides assumes that there are at least three agents involved in a conflict: an agent providing a service (call it the server) and at least two other agents making conflicting requests of the server (call them the clients). In such a situation, the default strategy will have the server ask each of the clients to evaluate its own and the others' requests. If necessary, the server will continue to ask each of the clients "what-if" questions, asking them to propose new actions to extend given sequences of actions and to evaluate those actions and the resulting states until the server finds a sequence of actions that will satisfy all of the agents involved in the conflict (or until it gives up).

Figure 6 illustrates the game tree built as a result of this procedure for an application in which two planning agents are attempting to use a third agent in charge of controlling an arm to achieve the conflicting goals. Each edge in the graph represents an action request

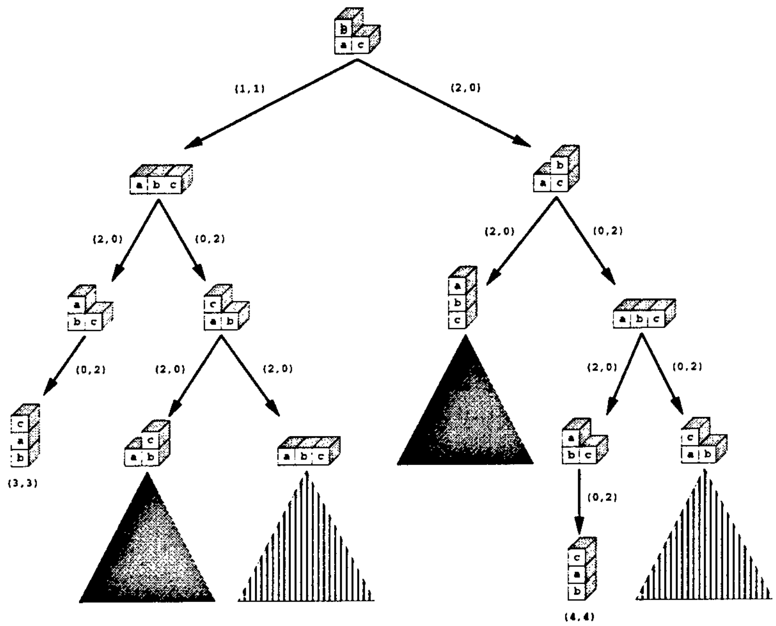

Figure 6: Result of default MASE conflict resolution strategy

from one of the planning agents for a given situation arising from the third agent's attempts to solve the conflict; it is labeled by both agents evaluation of that action. The resulting tree will be used by the third agent to execute an action sequence that will best satisfy both planners' requests.

3.2.2 MASE support for Temporal Reasoning

The MASE Agent also provides a general-purpose temporal reasoning engine that developers may rely upon when creating application-specific classes of agents. Initially it was provided to be used in conjunction with the conflict resolution strategy specified above, but it is available independently. The temporal reasoning engine allows agents to store time-stamped facts into a database, and then determine whether those facts are true at later points in time; the developers must provide application-specific routines to determine whether any two facts contradict each other.

The temporal engine provided by MASE does not perform generic deduction. A future release of MASE may provide a per-agent interface to an external Prolog-like reasoning engine instead, to provide user-definable mechanisms for temporal reasoning, and to provide agents the ability to do rule-based reasoning.

3.3 Using MASE to build applications

The design and implementation of systems of interacting agents using MASE involves the customization of the core Agent abstractions to add whatever application- and task-specific information each class of agent in the system will be responsible for maintaining. This is easily done in our implementation of MASE by deriving new classes from class Agent (or InteractiveAgent, if the interactive display capabilities are desirable) and adding to these new classes the application- or task-specific information.

In addition to any additional data that may be required, each class of agents derived from class Agent must provide its own methods for initializing, executing, and terminating any agents of that class created by the application. These three methods are responsible for the initialization and termination of each individual class of agent, as well as specifying the specific behavior that the agent will execute on every cycle.

3.3.1 A Sample Application

One of the applications built using MASE was designed to explore the behaviors of several different algorithms for routing and processing packages in a processing network. Each processing node in this network is capable of processing packages itself; it has a finite queue of incoming packages to be processed, and a set of pathways connecting it to neighboring nodes that it can use to forward packages from its own queue.

The application was designed to explore the behaviors of networks of processing nodes, where each node would be controlled by some of the following algorithms:

DumbDistributor A node following this algorithm would never forward packages to its neighbors;

InitRandom A node following this algorithm would initially choose randomly a single neighbor to forward packages to, and would then forward packages to it whenever it estimated its own queue was in danger of overflowing;

InitBest A node following this algorithm would initially choose the neighbor with the best chance of being able to handle packages forwarded to it (in the estimation of this node), and would then forward packages to it at need;

RandomDeliv A node following this algorithm would choose at random a neighbor to forward a package to, every time it chose to forward a package elsewhere;

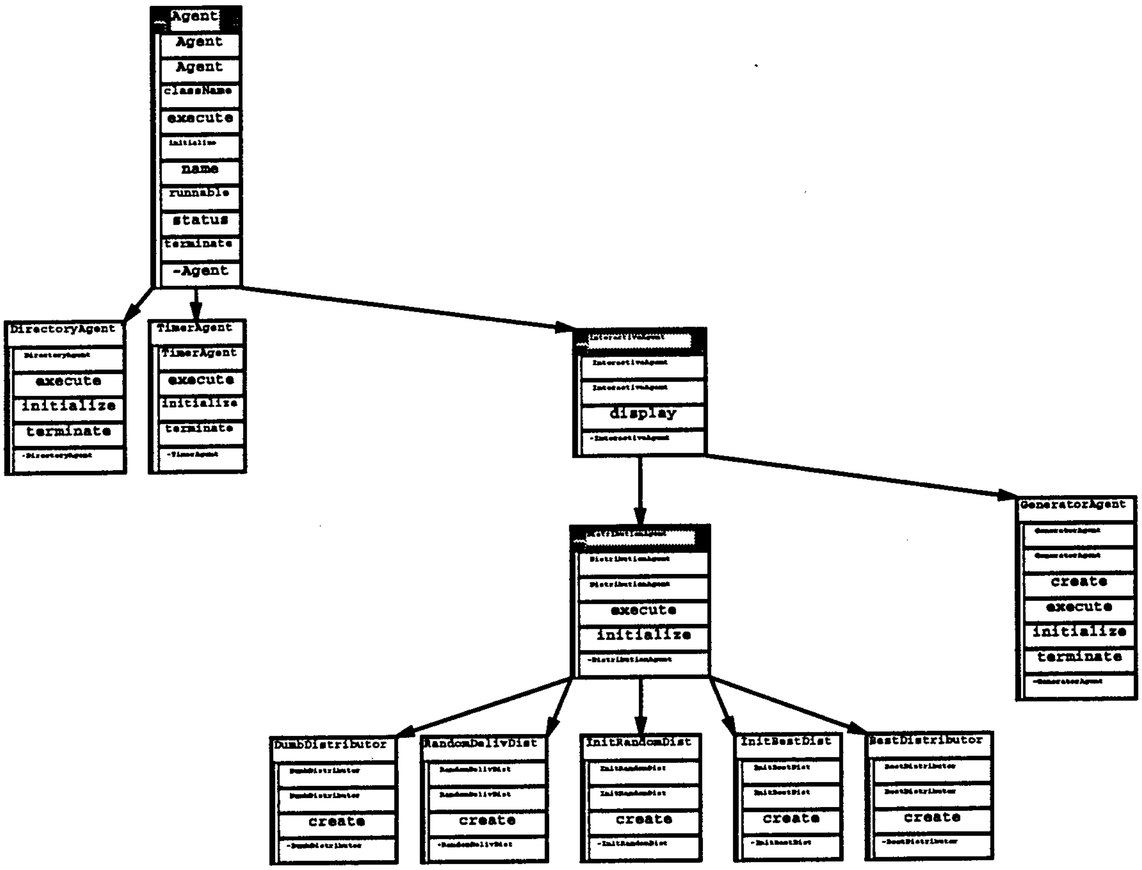

Figure 7: Agent Class Hierarchy

BestDistributor A node following this algorithm would keep track of the status of its neighbors, and would choose the neighbor most likely to be able to process a forwarded package, every time it chose to forward a package elsewhere.

We created a class DistributionAgent, to capture the information and behaviors relevant to the controllers in charge of all nodes in the network. This class was derived from InteractiveAgent, to gain access to all the support mechanisms provided by MASE. We then derived five classes from DistributionAgent, to capture the details of the five decision procedures described above. Figure 7 shows these relations graphically. Figure 8 shows several snapshots of a sample run of the resulting application, configured to study a network of 3 processing nodes plus a node to generate the packages to be processed.

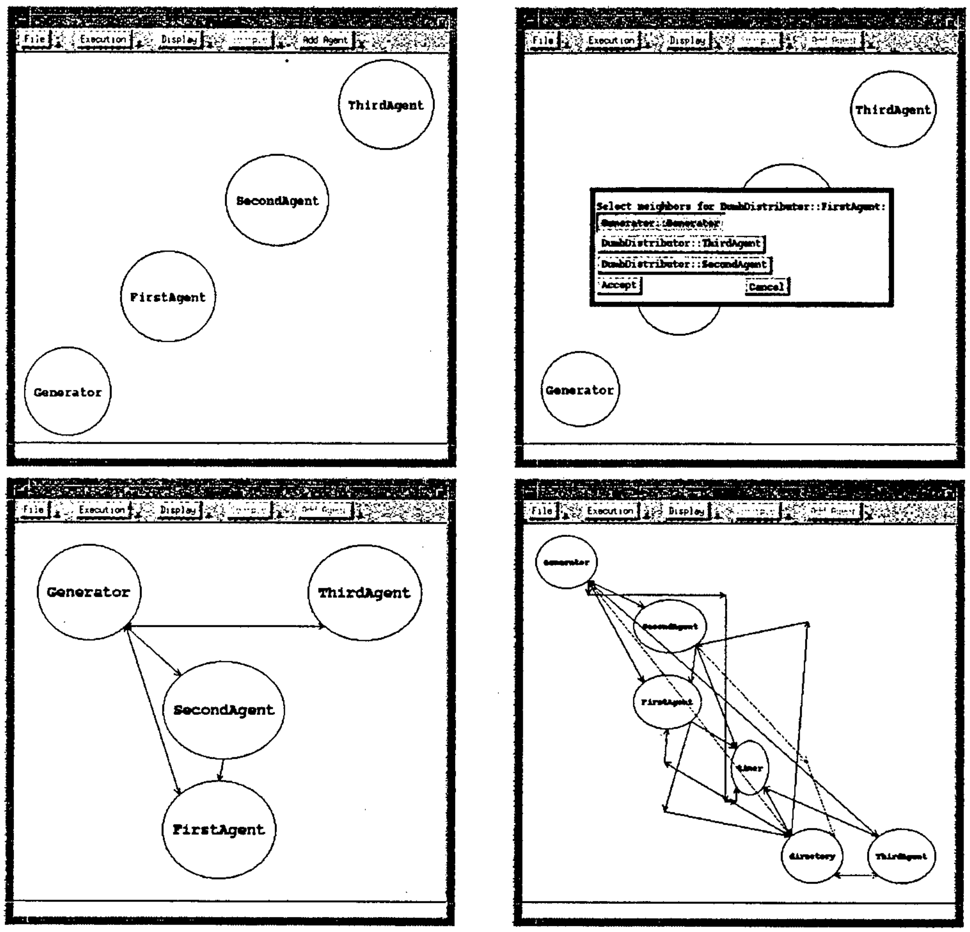

Figure 8: Some snapshots of a running application

3.4 Comparison of MASE to KQML

Knowledge Query Manipulation Language (KQML) is a protocol developed for the planning initiative to support high-level communication among independent knowledge-based software components. Both MASE and KQML provide layers of abstractions for inter-process communications that can be used to build sophisticated distributed applications.

In the case of KQML, emphasis was placed on the efficient transmission of the messages between processes over the underlying network; the set of communications abstractions provided by KQML are not much higher than the traditional operations provided by TCP/IP, concentrating on point-to-point message transfers using a single protocol (TCP/IP).

MASE (or more precisely, CoRaCLe) concentrates on providing very high-level communications abstractions, perhaps at the expense of transmission efficiency. At the protocol level, it provides the ability to transparently mix and match a variety of communications protocols (TCP/IP, RPC, shared-memory). In addition, it provides a more powerful model of inter-process communications, allowing processes to receive messages based on message contents, without requiring the creation of explicit communication links between sender and all the potential recipients of the message. MASE itself also provides abstractions for other purposes (like agent coordination and temporal reasoning) that fall outside the scope of KQML.

3.5 Availability

MASE and its communications subsystem, CoRaCLe, are available as libraries of C++ classes that implement all of the abstractions defined. In addition, a simple interface program is supplied that users can take advantage of to interact with a system that uses MASE. This program can also be used to provide MASE interfaces for systems that would find it awkward to invoke C++ code directly, such as systems built using Common Lisp. The solution involves the creation of a subprocess that will act as the Common Lisp system's proxy to the rest of the MASE universe. The Common Lisp system would then simply interact with this subprocess using the supported basic I/O operations. Sample code that implements this for Sun Common Lisp is included in the software distribution.

The software and documentation are available via anonymous FTP from the Internet host wilma.cs.brown.edu, in the directory ~ftp/pub. The two files "libMASE.tar.Z" and "libMASE.doc.tar.Z" contain the complete source code and the documentation for MASE, respectively. The two files "libCoRaCLe.tar.Z" and "libCoRaCLe.doc.tar.Z" contain the complete source code and the documentation for the communications subsystem (CoRaCLe) alone.

The MASE library was written for C++ version 3.0. The implementation of MASE relies on compiler support for the template mechanism introduced in that version of C++, so porting it to an older release may be a chore. It has been tested using the AT&T C++ compiler on Sun SparcStations running SunOS 4.1.3.

4 Decomposition Methods for Large MDPs

Having found MDPs appropriate for planning in stochastic domains in our work on envelope based methods, we continued to search for a general method for constructing abstract

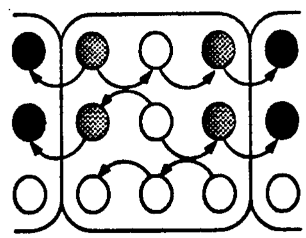

Figure 9: Boundary and periphery states

Markov decision processes for planning and scheduling in very large state spaces. In the latter part of the contract, our attention turned to various decomposition techniques. This section presents a very general approach which subsumes and considerably extends our envelope-based methods.

Let $M = (X, A, p, c)$ be a Markov decision process with finite state space X, actions A, state transition matrix p_{ij}, and cost matrix c_{ij}. Let P be any partition of X, $P = \{R_1, \ldots, R_n\}$ such that $X = \bigcup_{i=1}^{n} R_i$ and $R_i \cap R_j = \emptyset$ for all $i \neq j$. We refer to each $R \in P$ as an *aggregate* state.

Definition: The *boundary* of an aggregate state R (denoted Boundary(R)) is the set of all base-level states not in R but reachable in a single transition from a base-level state in R.

$$\text{Boundary}(R) = \{j | j \notin R \wedge \exists i \in R, p_{ij} > 0\}$$

Definition: The *periphery* of an aggregate state R (denoted Periphery(R)) is the set of all base-level states in R from which you can reach a state not in R in a single transition.

$$\text{Periphery}(R) = \{i | i \in R \wedge \exists j \notin R, p_{ij} > 0\}$$

In Figure 9 the boundary states are shaded light gray and the periphery states are shaded darker gray.

Definition: We say that aggregate state R in P is adjacent to aggregate state S in P (denoted $R \rightsquigarrow S$) just in case $\text{Boundary}(R) \cap S \neq \emptyset$ (see Figure 10).

We construct an abstract action for $R \rightsquigarrow S$ as follows.

1. Local state space $R \cup \text{Boundary}(R)$
2. Local transition matrix q_{ij}

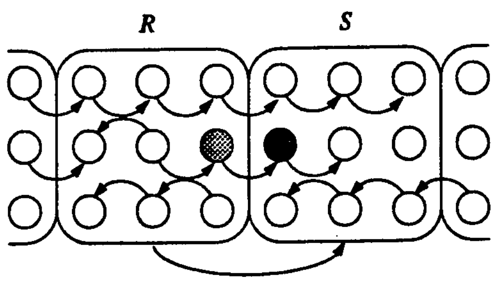

Figure : R is adjacent to S ($R \rightsquigarrow S$)

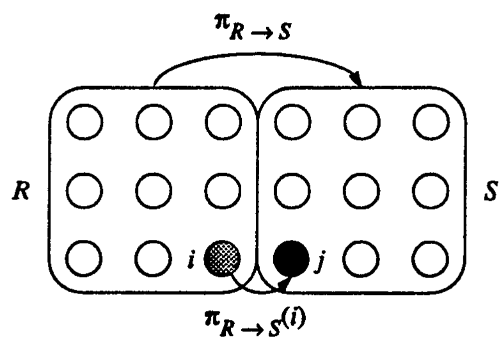

Figure 11: Abstract action for moving from R to S

- $q_{ij} = p_{ij}$ for $\forall i \in R$
- $q_{ij} = 1$ for $\forall i \in \text{Boundary}(R)$

3. Local cost matrix k_{ij}

   - $k_{ij} = c_{ij}$ for $\forall j \in R$
   - $k_{ij} = 0$ for $\forall j \in S$
   - $k_{ij} = 1$ for $\forall j \in \text{Boundary}(R) - S$

4. Compute the local policy $\pi_{R \to S}$ optimal for $(R \cup \text{Boundary}(R), A, q, k)$.

The set of all abstract actions is denoted

$$U = \{\pi_{R_i \to R_j} | i \neq j \wedge 1 \leq i, j \leq n\}$$

Figure 11 illustrates the difference between abstract actions defined as local policies ($\pi_{R \to S} \in U$) and base-level actions ($\pi_{R \to S}(i) \in A$).

The probability of ending up in T starting in R and following $\pi_{R\to S}$ is defined by

$$\begin{aligned}\phi_i &= \left[\sum_{j\in T\cup \text{Boundary}(R)} p_{ij}\right] + \sum_{j\in R} p_{ij}\phi_j \\ p'_{RT}(\pi_{R\to S}) &= \frac{1}{|\text{Periphery}(R)|}\left[\sum_{i\in \text{Periphery}(R)} \phi_i\right]\end{aligned}$$

where $p_{ij} = p_{ij}(\pi_{R\to S}(i))$.

The cost of ending up in T starting in R and following $\pi_{R\to S}$ is defined by

$$\begin{aligned}\xi_i &= \left[\sum_{j\in T\cup \text{Boundary}(R)} p_{ij}\right] + \sum_{j\in R} p_{ij}[1+\xi_j] \\ c'_{RT}(\pi_{R\to S}) &= \frac{1}{|\text{Periphery}(R)|}\left[\sum_{i\in \text{Periphery}(R)} \xi_i\right]\end{aligned}$$

where $c_{ij} = c_{ij}(\pi_{R\to S}(i))$.

The resulting abstract decision process is then defined by (P, U, p', c'). The approach described in [Dean *et al.*, 1994] and appearing in the special issue of *Artificial Intelligence* on planning represents a special case of the above general framework with two sets in P, one corresponding to the envelope and the other to everything outside the envelope. There is a rich literature based on related notions of decomposition and aggregation. Some of the classic papers that most of the techniques appear to be based on are listed below.

- Dantzig-Wolfe Decomposition [Dantzig & Wolfe, 1961],
- Krohn-Rhodes Decomposition [Krohn & Rhodes, 1965],
- Aggregation Disaggregation Techniques [Schweitzer, 1984], and
- Hierarchical Discrete Event Systems [Zhong & Wonham, 1990]

Now let's consider some potential problems with the above approach. We begin by reconsidering the method for defining an abstract action $\pi_{R\to S}$ for moving from aggregate state R to aggregate state S. Let $R \cup \text{Boundary}(R)$ be the local state space and $Q = S \cap \text{Boundary}(R)$ be the local target set. Let the transition probabilities be the same on R

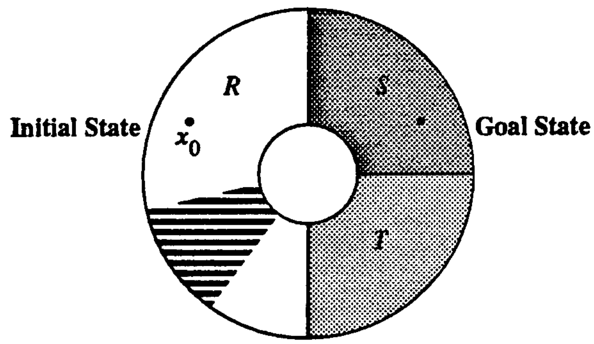

Figure 12: A problem with abstract actions

as in the base-level process and make each base-level state in Boundary(R) correspond to a sink (closed singleton set). Assign a cost of 1 to each state in R and a cost of 0 to each state in Q. What cost do we assign to $\tilde{Q} = \text{Boundary}(R) - Q$? If we assign states in $\tilde{Q}$ a cost of 0, then $\pi_{R \to S}$ may not move toward S any faster or more efficiently than toward any other aggregate state adjacent to R. If we assign states in $\tilde{Q}$ a cost of 1, then since the states in $\tilde{Q}$ are sinks the policy will avoid them in the face of any finite cost. If we assign states in $\tilde{Q}$ a one time (first passage) cost of λ, then we introduce a parameter that can affect the performance of $\pi_{R \to S}$ in potentially subtle ways.

To illustrate, consider the example shown in Figure 12. Suppose that the base-level state space is partitioned into three aggregate states $\{R, S, T\}$ as shown above. The shaded portion of R represents a portion of the state space through which it is difficult to pass. Starting from R the best strategy would be to head directly for S and hence a policy $\pi_{R \to S}$ seems desirable. However, in attempting to get to S from R it is possible (though unlikely) to pass through the shaded portion, in which case it is best to head for T and from there to S. The problem is that no abstract action of the form we have considered so far, in this case $\pi_{R \to S}$ or $\pi_{R \to T}$, is optimal.

So the immediate questions are as follows. Is it possible to establish an interesting worst case bound using the definition for abstract actions given above? Is there an alternative method for constructing abstract actions that would ensure better bounds? For example, perhaps there is some way of automatically tuning the λ parameters mentioned above. We will consider this latter suggestion in a little more detail.

Consider the situation shown in Figure 13 in which the aggregate state R is adjacent to two other aggregate states S and T. We construct the local decision process for R but this time we introduce two parameters, λ_{RS} and λ_{RT}, one for each region adjacent to R. We construct one abstract action π_R for R, rather than one action for each aggregate

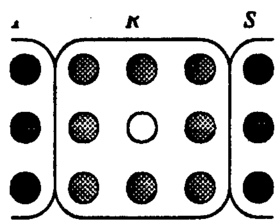

Figure 13: Adjacent aggregate states

state adjacent to R. We define the cost matrix as before except that $c_{ij} = \lambda_{RS}$ for $j \in R$ and $c_{ij} = \lambda_{RT}$ for $j \in T$. We want λ_{RS} to be an estimate of the cost of ending up in the aggregate state S and so we define λ_{RS} to be the average of $V_{\pi_S}(i)$ over all $i \in$ Boundary(S) (similarly for λ_{RT}). Now you can develop an iterative procedure for updating the λ values akin to policy iteration [Howard, 1960] or value iteration [2]. In the degenerate case (all singleton sets), abstract value iteration amounts to asynchronous dynamic programming. In the following, we present a sketch for abstract policy iteration.

Notation

- Π_i: abstract policy at the ith iteration
- π_R^i: local policy for R at the ith iteration
- λ_{RS}^i: λ parameter for R adjacent to S at the ith iteration
- ϵ: termination threshold $\epsilon > 0$

Algorithm

1. Set i to be 0.
2. Set $V_{\Pi_0}(R) = 0$ for all R in partition P.
3. Set $\lambda_{RS}^i = V_{\Pi_i}(S)$ for all R and S in partition P such that $R \rightsquigarrow S$.
4. Compute local policies π_R^i for all R [policy improvement].
5. Compute costs and transitions for the abstract decision process.
6. Compute $V_{\Pi_i}(S)$ for all S [value determination].
7. If $V_{\Pi_i}(S) < V_{\Pi_{i-1}}(S) + \epsilon$ for all S then quit [termination criterion].

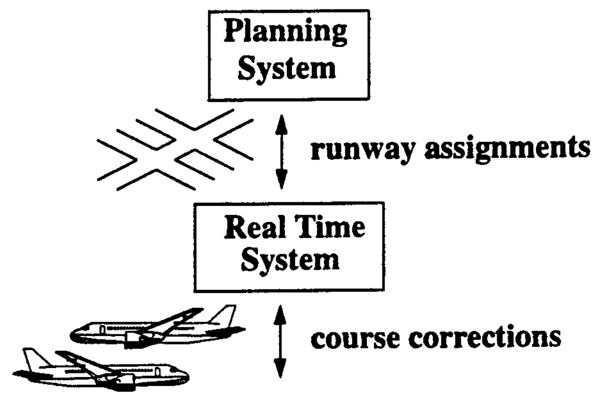

Figure 14: Embedded planning and control application

8. Set i to be $i + 1$ and go to Step 3.

Will the above algorithm converge and if so how quickly? Will specific classes of partitions converge more quickly? A technical report answering these questions and many more was produced during the last few month of the contract [11] and a paper summarizing our theoretical contributions will appear in IJCAI-95 [12].

5 Real-Time Planning and Problem Solving

For some time, we have been interested in the problem of how slow, high-level systems (*e.g.*, for planning and scheduling) might interact with faster, more reactive systems (*e.g.*, for real-time execution and monitoring). This issue arises in crisis-management systems, time-critical decision support, and any application in which computation time can delay decision making thereby affecting performance. For instance, in air traffic control, there is a combinatorial problem in scheduling the use of gates and runways so as to maximize airport throughput and there is also the problem of issuing course corrections to avoid collisions and ensure passenger safety and comfort (see Figure 14). The course corrections affect the feasibility of gate schedules and the gate schedules ultimately require course corrections. Passenger safety is the first priority but wherever passenger safety can be assured time should be spent improving throughput. We are interested in the design of systems that make the best use of the time available for decision making by explicitly accounting for the costs and benefits of computational delays.

We define *embedded planning* to be the problem of determining actions for a system embedded in an uncertain environment governed by dynamics outside of the system's con-

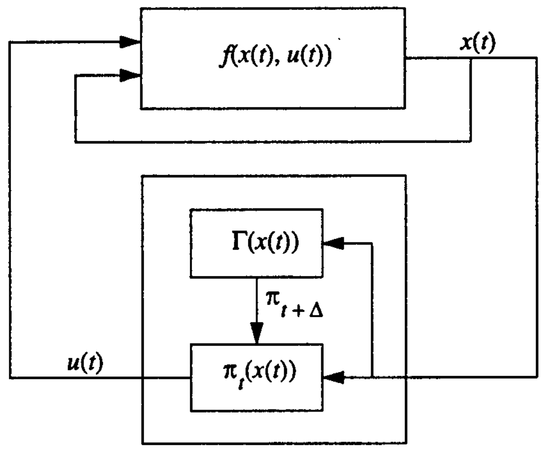

Figure 15: Model for embedded planning and control system

trol. Figure 15 provides the basic dynamical model for an embedded planning and control system. In Figure 15, $x(t)$ is the state at time t of the system we are seeking to control, $u(t)$ is the control action at time t taken by the composite planning and control system, and the function $f(x(t), u(t))$ determines the dynamics of the system that we are seeking to control. The control action is determined by a *policy* that can be executed by a real-time control system. To satisfy real-time constraints, the size of the policy must be bounded in accord with available computing resources. The policy π_t has a temporal index because it changes under the control of the planning component Γ. Due to the combinatorics involved in planning there is typically a delay Δ between when a state is observed and when a policy is available for execution. The dynamical system depicted in Figure 15 captures the essential properties of embedded planning and control systems: a changing environment that is not under the complete control of the planning system and delays in transmitting results between planning and control components. The system described in Section 2 represents a specific instance of the general model in shown Figure 15.

We are interested in this model for a number of reasons. First, it provides the basis for a mathematical model that supports detailed analyses of embedded planning and control systems. Second, we can generalize on the model to describe more complicated software systems consisting of many components. This more general model suggests an approach to specifying real-time systems that supports software reusability. We will consider each of these two aspects in turn.

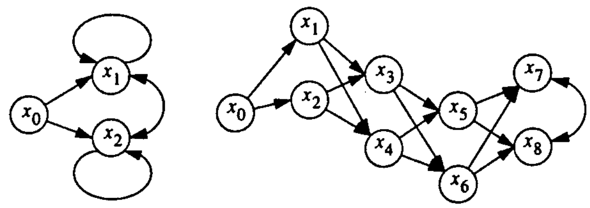

Figure 16: Two processes with uniform dispersion

5.1 Towards a Formal Theory of Embedded Systems

In planning problems that require predicting the behavior of complex processes, computational costs are typically a function of how such processes evolve over time. A slowly evolving process requires us to cope with small numbers of possible outcomes while a quickly evolving process can make prediction very difficult. Generally speaking, the greater the number of possible outcomes the more difficult decision making. It would be useful to develop some means of characterizing how processes evolve so that we can anticipate computational demands. In the following, we consider some simple properties of processes that serve to describe how they evolve over time. This analysis relates to the issues raised in connection with the generality of the techniques discussed in Section 2.

Figure 16 shows the state transition diagrams for two different stochastic processes. The number of states reachable from a given state is called the *dispersion rate* (or just the *dispersion*) of the state. If the dispersion is the same for all of the states of a given process, then the process is said to have *uniform dispersion.* The processes in Figure 16 are uniform with a dispersion rate of two.

In the process on the left in Figure 16, there are only two distinct states reachable following the initial state. In the process on the right in Figure 16, there are eight distinct states reachable from the initial state. The number of states reachable from a given state after n or less transitions is called the *diffusion characteristic for an n step lookahead.* Figure 17 illustrates graphically the diffusion characteristics for the two processes shown in Figure 16.

Suppose that in order to control a process it is enough to construct a table (corresponding to a policy) that maps states to actions to execute in those states. For the process on the left in Figure 16, the table would have to be of size two; for the process on the right in Figure 16, the table would have to be of size eight. If we make the obvious extrapolation for the process on the right in Figure 16, the table would have to be of size 2^n to account for all the possible states reachable in n transitions. A control system using a bounded policy will have to periodically compute a new table. To make optimal use of the time available,

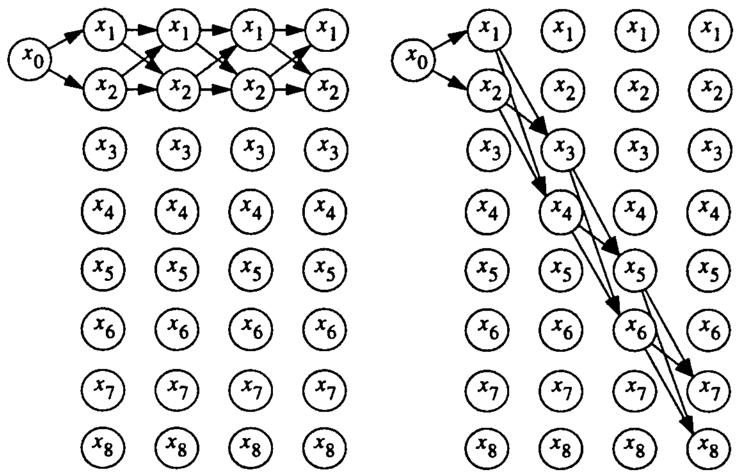

Figure 17: Processes with different diffusion characteristics

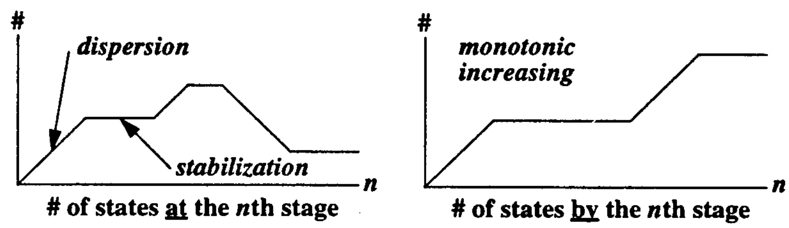

Figure 18: Dispersion and diffusion profiles for processes

the system will have to anticipate the costs of computing a new table which will depend on the properties of the process being controlled.

In anticipating the computational requirements for computing a new policy online, we would like to predict the behavior of the process over time. We can summarize aspects of this behavior using a *dispersion profile* that indicates the number of states the process might be in after the nth transition and a *diffusion profile* that indicates the number of states the process might have passed through by the nth transition. Figure 18 shows examples of each of these different types of profiles. Note that a diffusion profile is monotonic increasing.

Consider a particular instance of the problem of designing an embedded planning and control system that makes optimal use the time available for computing. Suppose that π_t corresponds to a table of size $\leq K$. Suppose that Γ takes three arguments: the current state $x(t)$, a lookahead parameter n corresponding to the number of transitions we looking forward, and a window size m corresponding to a target interval of time, $t+n$ to $t+n+m$. Γ

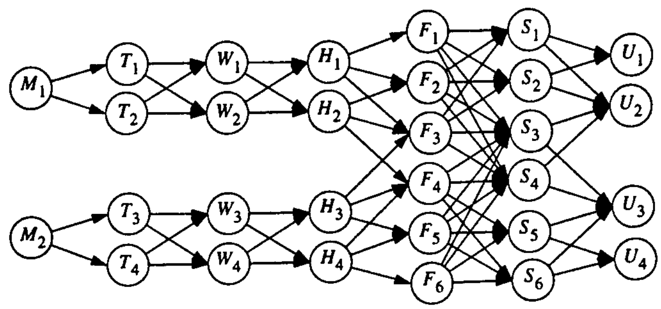

Figure 19: Process with periodic dispersion

computes the policies π_{t+n} through π_{t+n+m}, for the times $t+n$ through $t+n+m$, respectively, subject to the table-size bound K under some optimality criterion. We assume that the computation time required by Γ for a given, x, n, and m, is determined by a function $h(x,n,m)$. $h(x,n,m)$ determines the delay Δ between when Γ is invoked and when its resulting policy is available for use as given in the model of Figure 15. The design task is to determine a strategy for invoking Γ in particular states with particular lookaheads and window sizes. Such a rule could invoke Γ and then terminate it before completion. The objective is to ensure that if the system arrives in a state at time t then $x(t)$ will be in the table defined by π_t.

The above discussion just begins to describe the types of problems and formal analyses that are possible in describing embedded planning and control systems. Most the problems that involve the design of such systems are computationally complex and it is necessary to exploit additional structure in the problem to make progress. One source of structure comes from the fact that most of the processes that we are concerned with exhibit predictable periodic behavior. For instance, consider the task faced by medical professionals in anticipating injuries and stocking supplies for an emergency room. Early in the week, changes occur slowly and there are not too many states that the system could transition to. As the weekend nears, the number of states that the system could transition to grows. Figure 19 depicts a process with a period of one week in which there are two possible Monday states, four each of Tuesday, Wednesday, Thursday, and Sunday states, and six each of Friday and Saturday states. Given that different states require different supplies and obtaining supplies takes time and given that only a limited number of supplies can be stocked, the problem is to determine when the emergency room should place their order for supplies and for which states it should prepare. This simple example has counterparts in transportation scheduling, vehicle routing, and other difficult problems with real-time constraints.

5.2 Software Specifications for Embedded Systems

Coping with combinatorial problems in a real-time setting demands making concessions [7, 5]. Typically, concessions are in the form of algorithms that compute approximations. For instance, instead of computing an optimal solution to a scheduling problem, a system might compute a solution that is within a small factor of optimal but do so in a fraction of the time required for computing the optimal solution. In a complex, multicomponent software system, approximations will be passed along as arguments between subsystems. For instance, in a target tracking application, one component might compute the approximate location of a target, another the approximate orientation of a surveillance device, and a third component might take these two approximations and compute a trajectory for the surveillance device that approximates the optimal trajectory.

A useful specification for real-time software systems will include measures characterizing the quality of different approximations. A specification will have to describe different modes of interaction whereby one module can ask for an approximation of a particular quality and receive information about how long it will take to compute such an approximation. In general, we need ways of specifying interfaces that deal with approximations and computational delays.

The top diagram in Figure 20 shows the three components of an embedded scheduling system. Component #3 does the actual scheduling but it relies on the output of Components #1 and #2 which are responsible for gathering and interpreting data concerning the current availability of transportation assets and the readiness of crews. The more time available for checking on assets and crews the more precise the estimates that Components #1 and #2 provide. Similarly the more time allowed for scheduling the better the resulting schedules. The overall value is determined by the schedules which are dependent on the time allocated to scheduling and the precision of the asset and crew estimates. These dependencies and the corresponding value model are depicted in the diagram (called an *influence diagram* [19]) shown in the bottom of Figure 20. We are working on a model for specifying the embedded behavior of software modules. This model would cover more than just input/output behavior by addressing performance deficits due to delays, reduced bandwidth in communications, and accounting explicitly for the time spent in computation. The model will include standards for specifying delays, performance measures, and protocols for modules to communicate with one another regarding trading precision for time. Our model represents an extension and refinement of work in the data fusion community concerned with combining data interpretation routines [14]. To determine if one software module can be substituted for another we envision using Bayesian decision theory in the form of influence diagrams to assess the costs and benefits of the substitution. It is our contention that such a model is necessary to support real-time software development for a

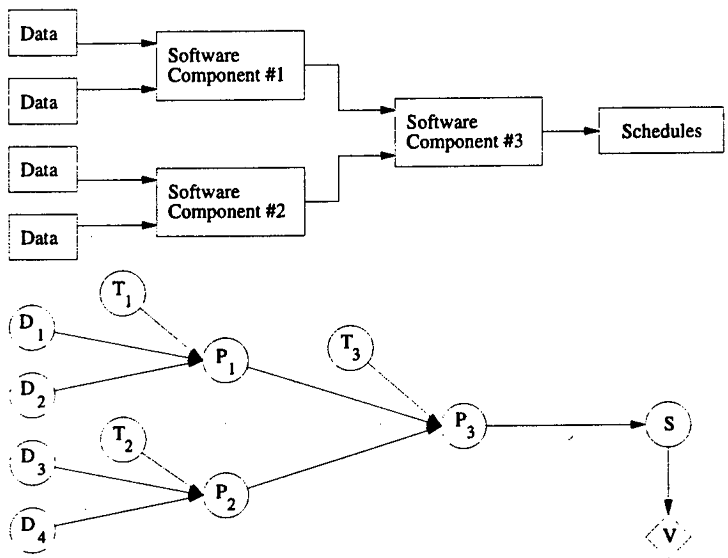

Figure 20: Software components for an embedded scheduling system and an influence diagram characterizing the dependencies between components and how they affect the expected value of the computed schedules. The D_i correspond to sources of data, the T_i to allocations of time, the P_i to normalized measures of precision, S to the resulting schedules, and V to a measure of value for the resulting schedules.

growing number of military and commercial applications.

References

[1] Andrews, Gregory R., Paradigms for Process Interaction in Distributed Programs, *ACM Computing Surveys*, **23**(1) (1991) 49–90.

[2] Bellman, Richard, *Dynamic Programming*, (Princeton University Press, 1957).

[3] Ben-Ari, M, Hoare, C. A. R., (Ed.), *Principles of Concurrent and Distributed Programming*, International Series in Computer Science, (Prentice-Hall, 1990).

[4] Bertsekas, Dimitri P., *Dynamic Programming: Deterministic and Stochastic Models*, (Prentice-Hall, Englewood Cliffs, N.J., 1987).

[5] Boddy, Mark and Dean, Thomas, Decision-Theoretic Deliberation Scheduling for Problem Solving in Time-Constrained Environments, *Artificial Intelligence*, **67**(2) (1994) 245–286.

[6] Chin, Roger S. and Chanson, Samuel T., Distributed Object-Based Programming Systems, *ACM Computing Surveys*, **23**(1) (1991) 91–124.

[7] Dean, Thomas and Boddy, Mark, An Analysis of Time-Dependent Planning, *Proceedings AAAI-88, St. Paul, Minnesota*, AAAI, 1988, 49–54.

[8] Dean, Thomas, Kaelbling, Leslie, Kirman, Jak, and Nicholson, Ann, Deliberation Scheduling for Time-Critical Sequential Decision Making, *Ninth Conference on Uncertainty in Artificial Intelligence, Washington, D.C.*, 1993, 309–316.

[9] Dean, Thomas, Kaelbling, Leslie, Kirman, Jak, and Nicholson, Ann, Planning With Deadlines in Stochastic Domains, *Proceedings AAAI-93, Washington, D.C.*, AAAI, 1993, 574–579.

[10] Dean, Thomas, Kaelbling, Leslie, Kirman, Jak, and Nicholson, Ann, Planning Under Time Constraints in Stochastic Domains, *Artificial Intelligence*, **76**(1-2) (1995) 35–74.

[11] Dean, Thomas and Lin, Shieu-Hong, *Decomposition Techniques for Planning in Stochastic Domains*, Technical Report CS-95-10, Brown University Department of Computer Science, 1995.

[12] Dean, Thomas and Lin, Shieu-Hong, Decomposition Techniques for Planning in Stochastic Domains, *Proceedings IJCAI 14, Montreal, Canada*, IJCAII, 1995, 1121–1127.

[13] Drummond, Mark and Bresina, John, Anytime Synthetic Projection: Maximizing the Probability of Goal Satisfaction, *Proceedings AAAI-90, Boston, Massachusetts*, AAAI, 1990, 138–144.

[14] Durrant-Whyte, Hugh F., *Integration, Coordination and Control of Multi-Sensor Robot Systems*, (Kluwer, Boston, Massachusetts, 1988).

[15] Gasser, Les, Braganza, Carl, and Nava, Herman, MACE: A Flexible Testbed for Distributed AI Research, Huhns, Michael N., (Ed.), *Distributed Artificial Intelligence*, (Morgan Kauffman, Los Altos, CA, 1987).

[16] Green, Peter E., AF: A Framework for Real-Time Distributed Cooperative Problem Solving, Huhns, Michael N., (Ed.), *Distributed Artificial Intelligence*, (Morgan Kauffman, Los Altos, CA, 1987).

[17] Hayes-Roth, Barbara, A Blackboard Architecture for Control, *Artificial Intelligence*, **26** (1985) 251–321.

[18] Howard, Ronald A., *Dynamic Programming and Markov Processes*, (MIT Press, Cambridge, Massachusetts, 1960).

[19] Howard, Ronald A. and Matheson, James E., Influence Diagrams, Howard, Ronald A. and Matheson, James E., (Eds.), *The Principles and Applications of Decision Analysis*, (Strategic Decisions Group, Menlo Park, CA 94025, 1984).

[20] Kemeny, J. G. and Snell, J. L., *Finite Markov Chains*, (D. Van Nostrand, New York, 1960).

[21] Kraus, Sarit and Wilkenfeld, Jonathan, *Negotiations Over Time in a Multi-Agent Environment*, Technical Report UMIACS-TR-91-51, Institute for Advanced Computer Studies, University of Maryland, April 1991.

[22] Reiss, Steven P. and Stasko, John T., The Brown Workstation Environment: A User Interface design Toolkit, *Preprints of the IFIP WG2.7 Working Conference on Engineering for Human-Computer Interaction*, August 1989.

[23] Rosenschein, Jeffrey S. and Genesereth, Michael R., Deals Among Rational Agents, *Proceedings IJCAI 9, Los Angeles, California*, 1985, 91–99.

[24] Zlotkin, Gilad and Rosenschein, Jeffrey S., Negotiation and Task Sharing Among Autonomous Agents in Cooperative Domains, *Proceedings IJCAI 11, Detroit, Michigan*, IJCAII, 1989, 912–917.

[25] Zlotkin, Gilad and Rosenschein, Jeffrey S., Negotiation and Conflict Resolution in Non-Cooperative Domains, *Proceedings AAAI-90, Boston, Massachusetts*, AAAI, 1990, 100–105.

☆U.S. GOVERNMENT PRINTING OFFICE: 1998-610-130-61163

DISTRIBUTION LIST

addresses	number of copies
WAYNE BOSCO RL/C3CA 525 BROOKS ROAD ROME NY 13441-4505	5
BROWN UNIVERSITY COMPUTER SCIENCE DEPARTMENT BOX 1910 PROVIDENCE, RI 02912	5
ROME LABORATORY/SUL TECHNICAL LIBRARY 26 ELECTRONIC PKY ROME NY 13441-4514	1
ATTENTION: DTIC-OCC DEFENSE TECHNICAL INFO CENTER 8725 JOHN J. KINGMAN ROAD, STE 0944 FT. BELVOIR, VA 22060-6218	2
ADVANCED RESEARCH PROJECTS AGENCY 3701 NORTH FAIRFAX DRIVE ARLINGTON VA 22203-1714	1
DR JAMES ALLEN COMPUTER SCIENCE DEPT/BLDG RM 732 UNIV OF ROCHESTER WILSON BLVD ROCHESTER NY 14627	1
DR YIGAL ARENS USC-ISI 4676 ADMIRALTY WAY MARINA DEL RAY CA 90292	1
DR MARIE A. BIENKOWSKI SRI INTERNATIONAL 333 RAVENSWOOD AVE/EK 337 MENLO PRK CA 94025	1

DR MARK S. BODDY 1
HONEYWELL SYSTEMS & RSCH CENTER
3660 TECHNOLOGY DRIVE
MINNEAPOLIS MN 55418

DR MARK BURSTEIN 1
BBN SYSTEMS & TECHNOLOGIES
10 MOULTON STREET
CAMBRIDGE MA 02138

DR GREGG COLLINS 1
INST FOR LEARNING SCIENCES
1890 MAPLE AVE
EVANSTON IL 60201

MS. LAURA DAVIS 1
CODE 5510
NAVY CTR FOR APPLIED RES IN AI
NAVAL RESEARCH LABORATORY
WASH DC 20375-5337

DR THOMAS L. DEAN 1
BROWN UNIVERSITY
DEPT OF COMPUTER SCIENCE
P.O. BOX 1910
PROVIDENCE RI 02912

DR PAUL R. COHEN 1
UNIV OF MASSACHUSETTS
COINS DEPT
LEDERLE GRC
AMHERST MA 01003

DR JON DOYLE 1
LABORATORY FOR COMPUTER SCIENCE
MASS INSTITUTE OF TECHNOLOGY
545 TECHNOLOGY SQUARE
CAMBRIDGE MA 02139

MR. STU DRAPER 1
MITRE
EAGLE CENTER 3, SUITE 8
O'FALLON IL 62269

DR MICHAEL FEHLING 1
STANFORD UNIVERSITY
ENGINEERING ECO SYSTEMS
STANFORD CA 94305

RICK HAYES-ROTH 1
CIMFLEX-TEKNOWLEDGE
1810 EMBARCADERO RD
PALO ALTO CA 94303

DR JIM HENDLER 1
UNIV OF MARYLAND
DEPT OF COMPUTER SCIENCE
COLLEGE PARK MD 20742

MR. MORTON A. HIRSCHBERG, DIRECTOR 1
US ARMY RESEARCH LABORATORY
ATTN; AMSRL-CI-CB
ABERDEEN PROVING GROUND MD
21005-5066

MR. MARK A. HOFFMAN 1
ISX CORPORATION
1165 NORTHCHASE PARKWAY
MARIETTA GA 30067

DR RON LARSEN 1
NAVAL CMD, CONTROL & OCEAN SUR CTR
RESEARCH, DEVELOP, TEST & EVAL DIV
CODE 444
SAN DIEGO CA 92152-5000

DR. ALAN MEYROWITZ 1
NAVAL RESEARCH LABORATORY/CODE 5510
4555 OVERLOOK AVE
WASH DC 20375

ALICE MULVEHILL 1
BBN
10 MOULTON STREET
CAMBRIDGE MA 02238

DR DREW MCDERMOTT 1
YALE COMPUTER SCIENCE DEPT
P.O. BOX 2158, YALE STATION
51 PROPSPECT STREET
MEW HAVEN CT 06520

DR DOUGLAS SMITH 1
KESTREL INSTITUTE
3260 HILLVIEW AVE
PALO ALTO CA 94304

DR. AUSTIN TATE 1
AI APPLICATIONS INSTITUTE
UNIV OF EDINBURGH
80 SOUTH BRIDGE
EDINBURGH EH1 1HN - SCOTLAND

DIRECTOR 1
DARPA/ITO
3701 N. FAIRFAX DR., 7TH FL
ARLINGTON VA 22209-1714

DR STEPHEN F. SMITH 1
ROBOTICS INSTITUTE/CMU
SCHENLEY PRK
PITTSBURGH PA 15213

DR JONATHAN P. STILLMAN 1
GENERAL ELECTRIC CRD
1 RIVER RD, RM K1-5C31A
P. O. BOX 8
SCHENECTADY NY 12345

DR EDWARD C.T. WALKER 1
BBN SYSTEMS & TECHNOLOGIES
10 MOULTON STREET
CAMBRIDGE MA 02138

DR BILL SWARTOUT 1
USC/ISI
4676 ADMIRALTY WAY
MARINA DEL RAY CA 90292

DR MATTHEW L. GINSBERG 1
CIRL, 1269
UNIVERSITY OF OREGON
EUGENE OR 97403

MR IRA GOLDSTEIN 1
OPEN SW FOUNDATION RESEARCH INST
ONE CAMBRIDGE CENTER
CAMBRIDGE MA 02142

MR JEFF GROSSMAN, CO 1
NCCOSC RDTE DIV 44
5370 SILVERGATE AVE, ROOM 1405
SAN DIEGO CA 92152-5146

DR ADELE E. HOWE 1
COMPUTER SCIENCE DEPT
COLORADO STATE UNIVERSITY
FORT COLLINS CO 80523

DR LESLIE PACK KAELBLING 1
COMPUTER SCIENCE DEPT
BROWN UNIVERSITY
PROVIDENCE RI 02912

DR SUBBARAO KAMBHAMPATI 1
DEPT OF COMPUTER SCIENCE
ARIZONA STATE UNIVERSITY
TEMPE AZ 85287-5406

DR MARK T. MAYBURY 1
ASSOCIATE DIRECTOR OF AI CENTER
ADVANCED INFO SYSTEMS TECH G041
MITRE CORP, BURLINGTON RD, MS K-329
BEDFORD MA 01730

MR DONALD P. MCKAY 1
PARAMAX/UNISYS
P O BOX 517
PAOLI PA 19301

DR MARTHA E POLLACK 1
DEPT OF COMPUTER SCIENCE
UNIVERSITY OF PITTSBURGH
PITTSBURGH PA 15260

DR MANUELA VELOSO 1
CARNEGIE MELLON UNIVERSITY
SCHOOL OF COMPUTER SCIENCE
PITTSBURGH PA 15213-3891

DR DAN WELD 1
DEPT OF COMPUTER SCIENCE & ENG
MAIL STOP FR-35
UNIVERSITY OF WASHINGTON
SEATTLE WA 98195

DR TOM GARVEY 1
DARPA/ISO
3701 NORTH FAIRFAX DRIVE
ARLINGTON VA 22203-1714

DIRECTOR
DARPA/ISO
3701 NORTH FAIRFAX DRIVE
ARLINGTON VA 22203-1714 1

OFFICE OF THE CHIEF OF NAVAL RSCH
ATTN: MR PAUL QUINN
CODE 311
800 N. QUINCY STREET
ARLINGTON VA 22217 1

DR GEORGE FERGUSON
UNIVERSITY OF ROCHESTER
COMPUTER STUDIES BLDG, RM 732
WILSON BLVD
ROCHESTER NY 14627 1

DR STEVE HANKS
DEPT OF COMPUTER SCIENCE & ENG'G
UNIVERSITY OF WASHINGTON
SEATTLE WA 98195 1

DR ADNAN DARWICHE
INFORMATION & DECISION SCIENCES
ROCKWELL INT'L SCIENCE CENTER
1049 CAMINO DOS RIOS
THOUSAND OAKS CA 91360 1

ROBERT J. KRUCHTEN
HQ AMC/SCA
203 W LOSEY ST, SUITE 1016
SCOTT AFB IL 62225-5223 1

DR. MAREK RUSINKIEWICZ
MICROELECTRONCS & COMPUTER TECH
3500 WEST BALCONES CENTER DRIVE
AUSTIN, TX 78759-6509 1

MAJOR DOUGLAS DYER/ISO
DEFENSE ADVANCED PROJECT AGENCY
3701 NORTH FAIRFAX DRIVE
ARLINGTON, VA 22203-1714 1

DR. STEVE LITTLE
MAYA DESIGN GROUP
2100 WHARTON STREET S&E 702
PITTSBURGH, PA 15203-1944 1

1

NEAL GLASSMAN
AFOSR
110 DUNCAN AVENUE
BOLLING AFB, WASHINGTON, D.C.
29332