Toward Ubiquitous Satisficing Agent Control *†

Thomas Wagner Victor Lesser

Computer Science Department
University of Massachusetts
Amherst, MA 01003
Email: {wagner,lesser}@cs.umass.edu

UMass Computer Science Technical Report 1997-61

December 20, 1997

Abstract

The dynamics and unpredictability of open environments pose a challenge to agent development. Agents operating in these environments must be able to adapt processing to the available resources and the current problem solving context. The ability to satisfice ubiquitously, in all aspects of agent problem solving, is the key to existing in these environments. Our work on satisficing agent control problem solving, i.e., multi-agent coordination and agent scheduling, serves as the foundation for our current focus on a holistic approach to satisficing agent control.

1 Introduction

With the advent of open computing environments adaptability in software applications is critical. Since open environments are less predictable, applications must be able to adapt their processing to the available resources (hardware, software, time, money, databases, etc.) and the different goal criteria set by different clients. In agent based systems, where software applications are persistent, autonomous, goal oriented, and function in real-time, this requirement is doubly important. In this context agents must be able to reason about their local problem solving activities, interact with other agents, plan a course of action, and carry out the actions in the face of limited resources and uncertainty about action outcomes and the actions of other agents, all in real-time and within real resource and cost constraints. Furthermore, in dynamic open environments new tasks can be generated by existing or new agents at any time, thus an agent's deliberation must be interleaved with execution – rescheduling, replanning, and recoordination with other agents is the norm, not the exception. To make matters even more difficult the planning and scheduling tasks are generally non-trivial, requiring either exponential work or, in practice, a sophisticated solution scheme that controls the algorithmic complexity and makes agent control a tractable problem.

Our most recent work in agent control, namely Design-to-Criteria [11, 9] scheduling and GPGP [2, 6] multi-agent coordination is directed at addressing such agent control issues. We believe that the key to operating in these open environments is the ability to satisfice ubiquitously, and dynamically, to adjust problem solving to the changing environment and problem solving context. One view of this idea, that also appears in our work, is the notion of resource-bounded-reasoning, i.e., solving a problem in different amounts of time or with different allotments of computational resources, perhaps trading-off solution quality and resource consumption. This is classically illustrated by an anytime [12] curve and typically translates into parameterizing the size of the solution space that is searched during problem solving (or the level of abstraction at which problem solving occurs). However, this view is only part of the picture. Satisficing, and the ability to satisfice, is intertwined

* **A version of this paper appears in the Proceedings of the 1998 AAAI Symposium on Satisficing Models.**

† This material is based upon work supported by the National Science Foundation under Grant No. IRI-9523419 and the Department of the Navy and Office of the Chief of Naval Research, under Grant No. N00014-95-1-1198. The content of the information does not necessarily reflect the position or the policy of the Government or the National Science Foundation and no official endorsement should be inferred.

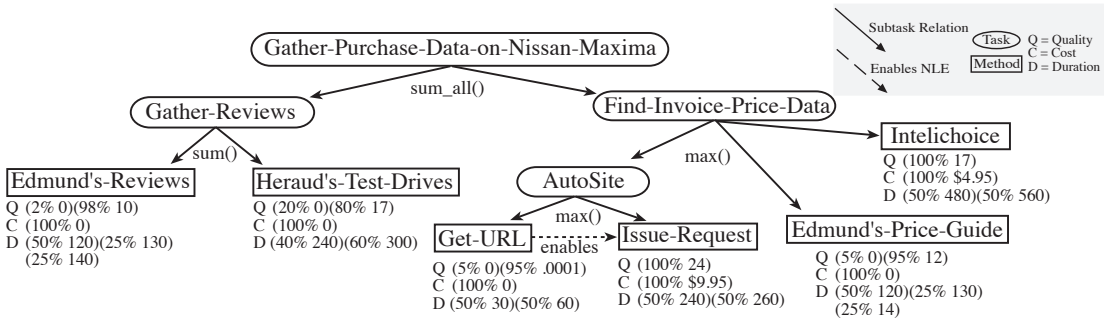

Figure 1: TÆMS Task Structure for Gathering Auto Purchase Information

with the ability to approach problem solving from a flexible perspective, to attack problems using different techniques or using different problem solving parameter settings, or consuming different resources or quantities of resources. Flexibility enables satisficing behavior. The existence of choices or alternatives begets the ability to perform differently in different circumstances – this is why the ability to satisfice is important. In different circumstances, different problem solving behaviors are appropriate. The choice of which satisficing behavior to employ is situation specific, i.e., dependent on the problem solving context at hand. These three concepts, *satisficing*, *flexibility*, and *situation specificity* are all interrelated and the boundary between these concepts blur when they are examined closely. Through our work, we have come to see and partially understand the relationships between these concepts and the importance of addressing all of these notions when building agent control systems. Understanding the full ramifications of these concepts is taking the problem one step further. We will return to this issue in the architecture section, but clearly satisficing on one aspect of problem solving normally has direct consequences to other aspects of problem solving, or problem solving that occurs downstream temporally.

Before delving into the details of our satisficing agent control work, let us ground further discussion in a high-level overview of the modeling framework used in these research projects. Our research focuses on a class of computational task structures where there are typically multiple goals and multiple different actions for performing a particular task, where each action has different statistical performance characteristics, and the outcomes of actions are highly uncertain. We model these problem solving activities using the TÆMS [2] domain-independent hierarchical task modeling framework. In TÆMS, primitive actions, called *methods*, are modeled statistically via discrete probability distributions in three dimensions, quality, cost, and duration. Quality is a deliberately abstract domain-independent concept that describes the contribution of a particular action to the overall problem solving objective. Duration describes the amount of time that a method will take to execute. Cost describes the financial or opportunity cost inherent in performing the action modeled by the method. It is no accident that problem solving activities are modeled in these four (uncertainty via the probability distributions) dimensions. Satisficing systems must operate in multiple dimensions to address the multi-dimensionality of the environments in which they exist. Problem solving issues are not typically limited to time, other items are also important, e.g., resources, financial costs, robustness, and precision. The ability to adapt problem solving for different multi-dimensional sets of goal criteria, amenable to future extension, is critical.

Returning to TÆMS, as with most hierarchical representations the high-level task is achieved by achieving some combination of its subtasks. Accordingly, since different methods have different quality, cost, duration, and certainty trade-offs, different solutions and partial solutions also have different characteristics and different trade-offs. Hard and soft interactions between tasks, called *NLEs* (non-local effects), are also represented in TÆMS and the effects of the interactions are reasoned about statistically during scheduling and coordination. TÆMS models are the grounding element and medium of exchange for Design-to-Criteria [9, 10] and Design-to-Time [4] scheduling, and and multi-agent [2] coordination research, and are being used in Cooperative-Information-Gathering [5], collaborative distributed design [3], distributed situation assessment [1], and surviveable systems [8] research projects.

A simplified example of a TÆMS task structure for gathering auto purchase information via the Web is shown in Figure 1. The oval nodes are tasks and the square nodes are methods. The top-level task is to *Gather-Purchase-Data-on-Nissan-Maxima* and it has two subtasks *Gather-Reviews* and *Find-Invoice-Price-Data*. The top-level task accumulates quality according to the *sum_all() quality accumulation function* (qaf)[1] so both of its subtasks

[1]Qafs define how a given task is achieved through its subtasks or methods. The *sum_all()* qaf means that all of the subtasks must

must be performed to satisfy the objective. The *Gather-Reviews* task has two methods, query *Edmund's-Reviews* and query *Heraud's-Test-Drives*. These methods are governed by a *sum()* qaf thus the power-set of the methods minus the empty set may be performed to achieve the tasks, i.e., *Edmund's* may be queried, *Heraud's* may be queried, or both may be queried. The *Find-Invoice-Price-Data* task has three subtasks, two of type method and one of type task, governed by the *max()* qaf which is analogous to an OR relationship. Note the decomposition of the obtain invoice via *AutoSite* task into two methods, one that locates the URL and one that issues the query. The *enables* NLE between the URL finding method and the query method, in conjunction with the low quality associated with the URL finding method, indicate that finding the URL is necessary for task achievement but that it contributes very little to achieving the task relative to the method that actually obtains the pricing report.

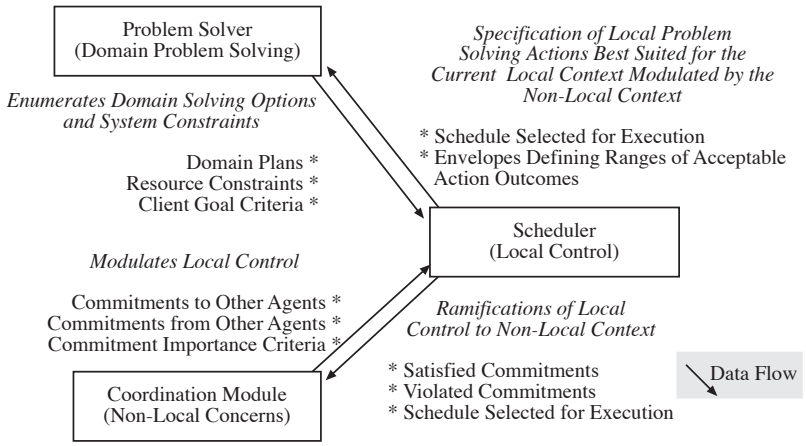

Figure 2: Key Agent Components

TÆMS models several important task features that facilitate a satisficing approach to problem solving. The existence of alternative ways to perform tasks gives TÆMS based problem solvers (schedulers, coordination algorithms) options about how to achieve tasks. Another feature of the model that supports satisficing is the statistical characterizations of primitive actions. The different statistical characteristics of alternative ways to achieve tasks gives us the ability to explore different possible solution paths, and consider the different trade-offs of each path, to find one that best satisfices to meet the current problem solving context. This is the TÆMS scheduling problem. For a given task structure and a given problem solving context, find a way (in real-time) to achieve the high level task that is in keeping with the context and the client's resource constraints. We will discuss our Design-to-Criteria satisficing scheduling work in more detail later in the paper. Modeling task interactions or non-local-effects is also an important feature of TÆMS. Explicit representation of both the quality of interactions and the quantified ramifications of interactions provides TÆMS clients with information that can be used to determine the relative importance of working to resolve interactions. From a scheduling perspective, this means reasoning about the benefits of leveraging task interactions relative to other constraints and to alternative ways of achieving the overall task. From a coordination perspective, agents can reason about the benefits of coordinating interactions with other agents versus the costs of such coordination. We will return to the issue of coordination shortly. Learning is also implicit in TÆMS. The probability distributions associated with primitive actions can be learned a priori or be refined over time via learning. Domain independence is also an important feature of TÆMS. The expense of custom designing agent components is serving to inhibit widespread agent development; moving toward a flexible and adaptable domain independent agent control system is critical for the future of agent systems. As we discuss in the architecture section, integrating domain independent agent control components is one of our primary near-term objectives.

In our work, agents generally exist in a multi-agent environment and each agent consists of three primary components: a domain problem solver, a scheduler that determines which domain actions to perform and in what sequence, and a coordination module that coordinates interactions with other agents. A high-level view of these three components is shown in Figure 2. In these systems, the domain problem solver describes domain problem solving options in the TÆMS language and emits the generated task structures along with goal criteria

be performed and that the task's quality is a sum of the qualities achieved by its subtasks.

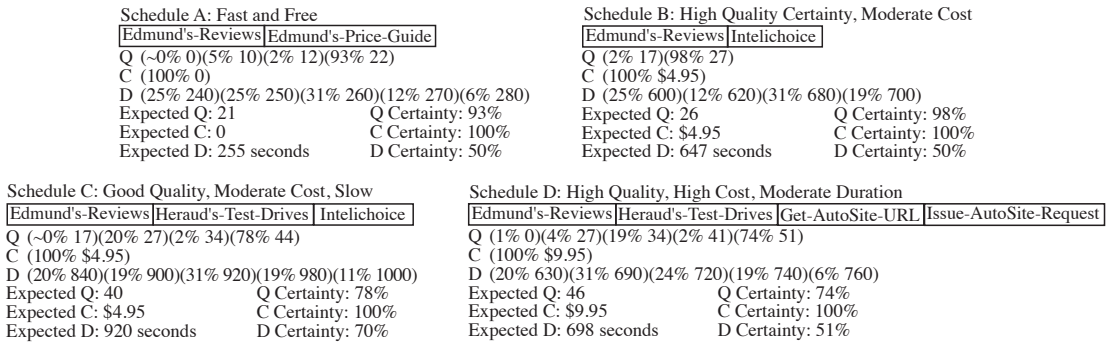

Figure 3: Four Satisficing Schedules

that specifies the desired quality, cost, duration, and certainty of a path through the task structure that achieves the high-level task. The scheduler inputs the task structures and goal criteria and constructs a schedule custom designed to achieve the task while adhering to the criteria specified by the problem solver. The coordination module works in conjunction with the scheduler to handle interactions with other agents by giving and receiving commitments to perform work at certain times. The importance of the non-local interactions is weighed against local problem solving concerns during scheduling; the resulting schedule is returned to the problem solver for execution. The schedule is annotated with performance envelopes and monitoring points define when rescheduling is necessary due to unexpected (or unprobable) execution results. Note that each agent has its own local scheduler and local coordination module – agents are autonomous and there is no centralized locale where all scheduling or coordination takes place. Agents can behave in a more aggregate fashion, but this is through the use of organization rather than fixed centralized control.

2 Design-to-Criteria Scheduling

The scheduler is the heart of satisficing agent control. An agent equipped with a Design-to-Criteria scheduler is able to adjust its problem solving activities to meet changes in resource requirements and to reason about the trade-offs of different courses of action. The central objective in Design-to-Criteria scheduling is to cope with the combinatorial explosion of possibilities while reasoning about a particular set of client goal criteria and the trade-offs presented by different solutions and partial solutions. In other words, scheduler client applications or users specify the design criteria and the scheduler *designs* a schedule to best meet the criteria, if possible given the task model. Figure 3 shows a set of satisficing schedules produced by the Design-to-Criteria scheduler, for the sample task structure (Figure 1), using four different sets of design criteria. Schedule A is constructed for a client interested in a fast, free, solution with any non-zero quality. Schedule B suits a conservative client who is interested primarily in certainty about quality achievement. Schedule C is designed for a client who wants high quality information, is willing to wait a long time for an answer, and is only willing to spend $5 on the search. Schedule D is meets the criteria of a client who wants the highest possible quality, is willing to spend $10, and wants the gathered data in 15 minutes or less.

The fundamental premise of our scheduling work is that *the goodness of a particular solution is entirely dependent on a particular client's complex objectives* and that *different client's have varying objectives.* Thus the scheduling process must not only consider the attribute trade-offs of different solutions, but must also do so dynamically. Furthermore, the scheduling process must be efficient – because of the inherent uncertainty in the domains, where actions may fail or have unexpected results, scheduling activities are interleaved with planning and execution. Thus scheduler inefficiencies are multiplied many times during a problem solving instance.

In general the upper-bound on the number of possible schedules for a task structure containing m methods is $\sum_{i=0}^{m} \binom{m}{i} i!$ and the $\omega(2^m)$ and $o(m^m)$ combinatorics of our scheduling problem precludes using exhaustive search techniques for finding optimal schedules. Design-to-Criteria copes with these explosive combinatorics by satisficing with respect to the goal criteria and with respect to searching the solution space. This satisficing dualism translates into four different techniques that Design-to-Criteria uses to reduce the search space and make the scheduling problem tractable:

Criteria-Directed Focusing The client's goal criteria is not simply used to select the "best" schedule for

execution, but is also leveraged to focus all processing activities on producing solutions and partial solutions that are most likely to meet the trade-offs and limits/thresholds defined by the criteria.

Approximation Schedule approximations, called *alternatives*, are used to provide an inexpensive, but coarse, overview of the schedule solution space.

Heuristic Decision Making The action ordering scheduling problem also suffers from large combinatorics. We cope with this complexity using a group of heuristics for action ordering. The heuristics take into consideration task interactions, deadlines, resource constraints, commitments made with other agents and so forth.

Heuristic Error Correction The use of approximation and heuristic decision making has a price – it is possible to create schedules that do not achieve the high-level task, or, achieve it poorly. A secondary set of improvement heuristics act as a safety net to catch the errors that may be correctable.

Design-to-Criteria thus copes with computational complexity by using the client goal criteria to focus processing, reasoning with schedule approximations rather than complete schedules, and using a heuristic, rather than exhaustive, scheduling approach. This methodology is effective because several aspects of the scheduling problem are soft and amenable to a satisficing approach. For example, portions of the client goal specification [9] express soft client objectives or soft constraints. Solutions often do not need to meet absolute requirements because clients cannot know *a priori* what types of solutions are possible for a given task structure due to the combinatorics. Similarly, soft task interactions also represent soft constraints that can be relaxed, i.e., they can be leveraged or not depending on the situation. Finally, though the TÆMS scheduling problem is more complex than many traditional scheduling problems because of its representation of multiple approaches for task achievement, it is also more flexible. If we view the scheduling activity as a search process, typically there is a neighborhood of solutions that will meet the client's goal criteria and the lack of exhaustive search, i.e., search by focused processing and approximation, does not necessitate scheduling failure. Several illustrations of Design-to-Criteria at work, and more algorithmic details, can be found in [11].

3 GPGP Multi-Agent Coordination

Satisficing in the GPGP (generalized partial global planning) multi-agent coordination work to date takes a different from than satisficing in Design-to-Criteria scheduling. In coordination, flexibility with respect to computational resources is derived from a modularized coordination approach that enables different modules, with different overhead costs, to be applied independently from one another. When resources are tight, only the most rudimentary coordination is used. When resources are less scarce, or the benefit of coordinating outweighs the cost of the coordination overhead, agents coordinate over "optional" interactions like soft interactions in TÆMS. The GPGP modularized coordination approach facilities a range of cost/benefit options for any problem solving episode as different subsets of coordination modules may be applied.

All coordination modules in GPGP coordinate through the use of *commitments*, that is inter-agent contracts to perform certain tasks by certain times. Commitments are made through the coordination mechanisms and considered by the scheduler when the local course of action is decided. GPGP defines the following coordination mechanisms (for the formal details see [2]):

Share Non-Local Views This most basic coordination mechanism handles the exchange of local views between agents and the detection of task interactions.

Communicate Results This coordination mechanism handles communicating the results of method execution to other agents, at various levels of detail and coverage as defined by different communication policies.

Avoid Redundancy This mechanism deals with detected redundancy by committing an agent at random to execute the redundant method in question.

Handle Hard Task Relationships - The *enables* NLE pictured in Figure 1 denotes a hard task relationship. This coordination mechanism deals with such hard, non-optional, task interactions by committing the predecessors of the *enables* to perform the task by a certain deadline.

Handle Soft Task Relationships Soft task interactions, unlike hard interactions like *enables*, are optional. When employed, this coordination mechanism attempts to form commitments on the predecessors of the soft interactions to perform the methods in question by a certain deadline.

As mentioned above, the GPGP coordination module modulates local control by placing constraints, commitments, on the local scheduler. The commitments generally fall into three categories: 1) *deadline* commitments are contracts to perform work by a certain deadline, 2) *earliest start time* commitments are agreements to hold-off on performing certain tasks until a certain time has passed, and 3) *do* commitments are agreements to perform certain tasks without a particular time constraint.

GPGP achieves flexibility through a modularized approach to coordination. The question then becomes which combinations of mechanisms are most effective. Empirical analysis [2] has shown that no single coordination mechanism, or sets of mechanisms, is most effective for all situations. Instead, as shown in Prassad and Lesser [6], coordination is best viewed as a situation specific activity; which set of mechanisms is most effective is dependent on the types of tasks in question and the overhead costs of communications. In different environments, different mechanisms are most effective.

4 The Future: A Holistic Satisficing Agent Architecture

Both GPGP and Design-to-Criteria are able to adapt processing to a given situation. GPGP uses a modularized approach to provide different degrees of coordination, each with its own overhead costs and benefits to problem solving. Design-to-Criteria uses a satisficing methodology to control the combinatorics and to produce schedules in real-time. This illustrates an important point; satisficing can take many forms, e.g., performing less search (Design-to-Criteria), or using less context to make decisions (GPGP), or working from default assumptions (situation specific GPGP). In fact, the implementation of GPGP is itself an illustration of satisficing by sacrificing context. GPGP forms commitments between agents through a one-shot mechanism where two agents form an agreement about task performance independently of the other agents in the system. This mechanism reduces the overhead required for coordination at the expense of making the commitments without understanding more of the group context. In contrast, if the commitment forming context is broadened to include a group of problem solving agents, to support a multi-step negotiation process, the overhead of commitment formulation will increase but so will the amount of information used to make commitment decisions.

In addition to meeting deadlines, Design-to-Criteria also builds custom schedules to suit a particular client's multi-dimensional goal criteria. This latter Design-to-Criteria feature represents one next step in the evolution of GPGP and our multi-agent coordination work. Currently, though GPGP provides a modularized toolbox of coordination mechanisms, it does not reason about which mechanisms to use to meet a particular client's needs (or goal criteria). The work in learning which coordination algorithms to use in particular situations is a step in the right direction, but this work also does not address meeting dynamic multi-dimensional goal criteria. Like Design-to-Criteria, GPGP must be able to adapt its processing automatically to a given problem solving context and resource constraints.

Despite Design-to-Criteria's strengths, neither aspect of our agent control work is complete. Neither system accounts for the actual cost of the control problem solving activity. In other words, though the scheduler copes with hard combinatorics to produce schedules in interactive time, it does not account for the resource costs of the scheduling process itself, i.e., Design-to-Criteria is fast, but it doesn't reason about the time required to find a good schedule and account for this time, and adjust its activities accordingly. Through the focusing mechanism it can adjust its own problem solving to different resource requirements, but right now, the focus is defined by the client or hardwired defaults.

The right approach for coordination and scheduling, and overall agent control, is to account for the both the control and domain problem solving costs, to reason about resource allocations and the cost/benefits of such allocations, and to control the agent accordingly. The benefits of spending time doing further coordination or more refined scheduling versus time spent problem solving must all be compared and reasoned about. This meta analysis [7] approach to agent control in conjunction with the satisficing abilities of Design-to-Criteria and GPGP, will result in agents being able to satisfice all activities and adapt to a wide range of dynamic problem solving environments.

However, this is an oversimplification of the problem. It is important to note that the issue is not to simply determine a resource allocation to control and domain problem solving, but to grapple with the downstream ramifications of the allocations and satisficing behaviors employed or considered. Control problem solving and domain problem solving are interdependent activities. Consider a case where time is allocated between a domain problem solver and a scheduler, and the domain problem solver faithfully describes its problem solving options

in a TÆMS task structure and then asks the scheduler to determine a course of action. If the scheduler satisfices, to meet its time resource constraint, it will do so by exploring a smaller part of the schedule solution space. One possible outcome of this is that the scheduler will fail to generate a very good solution and will instead generate and return a mediocre solution. The domain problem solver will then execute the schedule and obtain *lower* quality results than it might have if less time had been allocated to domain problem solving and more time had been allocated to scheduling. In this example, the downstream effect of control satisficing was not generating a good result in the time alloted. In other cases, the ramifications range from the introduction of additional uncertainty to being unable to find any solution because one of the components was over constrained.

The previous example is cast in terms of time allocations, but as mentioned earlier, satisficing can be in many dimensions like precision, robustness, and cost, and it can be in terms of multiple dimensions simultaneously. This is what motivated our recent work in multi-dimensional goal criteria for Design-to-Criteria scheduling [9]. But even that work needs to be pushed to another level – satisficing to meet a particular set of multi-dimensional goal criteria is only part of the problem. In most problem solving situations, there are actually multiple sets of goal criteria, i.e., there is no single evaluation function. Consider a case where a client tells the scheduler to produce a schedule that will achieve the task while minimizing time and incurring no cost. If, for the given task structure, staying within these constraints means generating a very poor result, the client may want to modify the constraints and ask for another schedule, rather than execute the schedule that was returned for the first set of goal criteria. Agent control components, and the holistic agent control mechanism, must reason about multiple sets of goal criteria, or provide a means for negotiation or iterative refinement to generate a set of criteria that is suited for the problem solving context and the task at hand.

Given the complex issues surrounding the issue of satisficing, our goal of a satisficing domain-independent agent control architecture seems quite ambitious. The satisficing cohesion needed for this architecture will require that all components have the ability to meta analyze their problem solving activities and to estimate the *local* (as everything is intertwined) costs/benefits of resource allocations. From the scheduling and coordination perspective, this is an achievable objective. For the domain problem solver, meeting this requirement is application dependent. However, to the extent that the domain problem solver can enumerate its problem solving options as a TÆMS task structure, the scheduler can then reason about the domain problem solver's activities explicitly. In fact, we may move toward representing all major control and domain activities in TÆMS task structures, and even estimating and modeling (via nles) the downstream interactions of resource allocations, and analyzing it using the Design-to-Criteria scheduler. If this approach is used, we must then also account for the time required to perform the meta-analysis. However, for certain classes of task structures the scheduling analysis time is highly predictable. For large problem solving tasks, we also plan to explore changing the problem solving horizon, the distance between the current point in time and a point at which allocations will be reexamined.

Satisficing, flexibility, and multi-dimensional trade-off behavior, are critical for agents in complex dynamic open environments. Our future work is centered on a holistic, domain independent, agent control architecture designed to achieve these goals. It is our hope that with the architecture agent developers will be able wrap or embed their domain problem solvers and create satisficing and flexible multi-agent systems.

5 Acknowledgments

We would like to thank Professor Keith Decker, Professor Alan Garvey, Professor Norman Carver, and Dr. Nagendra Prassad, for their contributions to this ongoing work.

References

[1] Norman Carver and Victor Lesser. The DRESUN testbed for research in FA/C distributed situation assessment: Extensions to the model of external evidence. In *Proceedings of the First International Conference on Multiagent Systems*, June, 1995.

[2] Keith S. Decker. *Environment Centered Analysis and Design of Coordination Mechanisms.* PhD thesis, University of Massachusetts, 1995.

[3] Keith S. Decker and Victor R. Lesser. Coordination assistance for mixed human and computational agent systems. In *Proceedings of Concurrent Engineering 95*, pages 337–348, McLean, VA, 1995. Concurrent Technologies Corp. Also available as UMASS CS TR-95-31.

[4] Alan Garvey and Victor Lesser. Representing and scheduling satisficing tasks. In Swaminathan Natarajan, editor, *Imprecise and Approximate Computation*, pages 23–34. Kluwer Academic Publishers, Norwell, MA, 1995.

[5] Victor Lesser, Bryan Horling, Frank Klassner, Anita Raja, Thomas Wagner, and Shelley XQ. Zhang. Information Gathering as a Resource Bounded Interpretation Task. UMASS Department of Computer Science Technical Report TR-97-34, March, 1997.

[6] M.V. Nagendra Prasad and V.R. Lesser. Learning situation-specific coordination in generalized partial global planning. In *AAAI Spring Symposium on Adaptation, Co-evolution and Learning in Multiagent Systems*, Stanford, March 1996.

[7] Stuart Russell and Eric Wefald. Principles of metareasoning. *Artificial Intelligence*, 49, 1991.

[8] Regis Vincent, Bryan Horling, Thomas Wagner, and Victor Lesser. Survivability simulator for multi-agent adaptive coordination. In *Proceedings of the First International Conference on Web-Based Modeling and Simulation*, 1998. To appear. Also available as UMASS CS TR-1997-60.

[9] Thomas Wagner, Alan Garvey, and Victor Lesser. Complex Goal Criteria and Its Application in Design-to-Criteria Scheduling. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 294–301, July 1997. Also available as UMASS CS TR-1997-10.

[10] Thomas Wagner, Alan Garvey, and Victor Lesser. Leveraging Uncertainty in Design-to-Criteria Scheduling. UMASS Department of Computer Science Technical Report TR-97-11, January, 1997.

[11] Thomas Wagner, Alan Garvey, and Victor Lesser. Criteria-Directed Heuristic Task Scheduling. *International Journal of Approximate Reasoning, Special Issue on Scheduling*, 1998. To appear. Also available as UMASS CS TR-97-59.

[12] Shlomo Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–83, 1996.