Modeling Uncertainty and its Implications to Design-to-Criteria Scheduling*

Thomas A. Wagner Anita Raja Victor R. Lesser

Department of Computer Science
University of Massachusetts at Amherst
wagner@cs.umass.edu

December 8, 1999

Abstract

Open environments are characterized by their uncertainty and non-determinism. Agents need to adapt their task processing to available resources, deadlines, the goal criteria specified by the clients as well their current problem solving context in order to survive in these environments. If there were no resource constraints, then an optimal Markov Decision Process based policy would obviously be the best way to make scheduling decisions. However in real agent systems, these scheduling decisions have to be made in real-time making the off-line policy computationally infeasible in open environments. Design-to-Criteria scheduling is the soft real-time process of custom building a schedule to meet dynamic client goal criteria (including real-time deadlines), using a task model that describes alternate ways to achieve tasks and subtasks. Recent advances in Design-to-Criteria include the addition of uncertainty to the TÆMS computational task models analyzed by the scheduler and the incorporation of uncertainty in the scheduling process. Design-to-Criteria uses a heuristic approach for on-line scheduling of medium granularity tasks. It approximates the analysis used to generate an optimal policy by heuristically reasoning about the implications of uncertainty in task execution. Design-to-Criteria is related to Design-to-Time and flexible computation methodologies. The addition of uncertainty has also spawned a post-scheduling contingency analysis step that can be employed in deadline critical situations where the added computational cost is worth the expense. We describe the uncertainty representation and how it improves task models and the scheduling process, and provide empirical examples of uncertainty reduction in action. We also evaluate the performance of our heuristic-based approach using the performance of the policy generated by an optimal controller as the benchmark.

1 Introduction

It is paramount for agent-based systems to adapt to the dynamics of open environments. The agents need to adapt their task processing to available resources, deadlines, the goal criteria specified by the clients as well their current problem solving context in order to operate effectively in these environments. They also need to take into consideration the uncertainty inherent in task processing when deciding which tasks to execute and in what order. The scheduling problem is exponential and complicated by the existence of task interactions (primitive actions may not be independent) and by the existence of global and individual constraints on the primitive actions (individual deadlines, cost limits, earliest start times, and quality requirements).

*Effort sponsored by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory Air Force Materiel Command, USAF, under agreement number F30602-97-1-0249 and by the National Science Foundation under Grant number IIS-9812755 and number IRI-9523419. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), Air Force Research Laboratory, National Science Foundation, or the U.S. Government.

If there were no resource constraints, then an optimal policy would obviously be the best way to make these scheduling decisions. We can compute this optimal policy by formalizing the scheduling problem as a Markov Decision Process (MDP), a framework widely used for stochastic planning in artificial intelligence [8, 2, 33, 9]. A stochastic planning problem includes operators (or actions) that transform a state into one of several possible successor states, with each possible state transition occurring with some probability. A solution is usually cast in the form of a mapping from states to actions called a policy. A policy is executed by observing the current state and taking the action prescribed for it. Computing the optimal policy is an exponential problem and would involve off-line analysis. However in our research with deployed multi-agent systems, these scheduling decisions have to be made with respect to the current set of goals and their associated objective functions and resource constraints. This, in combination with a dynamic, open environment where agents may enter or leave the system, uncertainty is high, and information incomplete or approximate, the off-line production of an optimal policy via MDP is computationally infeasible.

In this paper, we present extensions to a heuristic approach for on-line scheduling of medium granularity tasks. It approximates the analysis used to generate an optimal MDP policy by heuristically reasoning about the implications of uncertainty in task execution using a less general but more compact representation of an MDP for representing possible task orderings. This heuristic approach called Design-to-Criteria (DTC) scheduling is a domain-independent soft real-time process of finding an execution path through a hierarchical task network such that the resultant schedule meets certain design criteria, such as real-time deadlines, cost limits, and quality preferences. Casting the language into an action-selecting-sequencing problem, the process is to select a subset of primitive actions from a set of candidate actions, and sequence them, so that the end result is an end-to-end schedule of an agent's activities that meets situation specific design criteria. In this model, when the action performance is not as expected, the scheduler is reinvoked and it reschedules in order to find the most appropriate sequence to complete the current goals. The combinatorics of the scheduling problem are controlled through the use of approximation, satisficing, goal-directed problem solving, and heuristics for action ordering, as discussed in [37]. We return to the issue of combinatorics in Section 4.

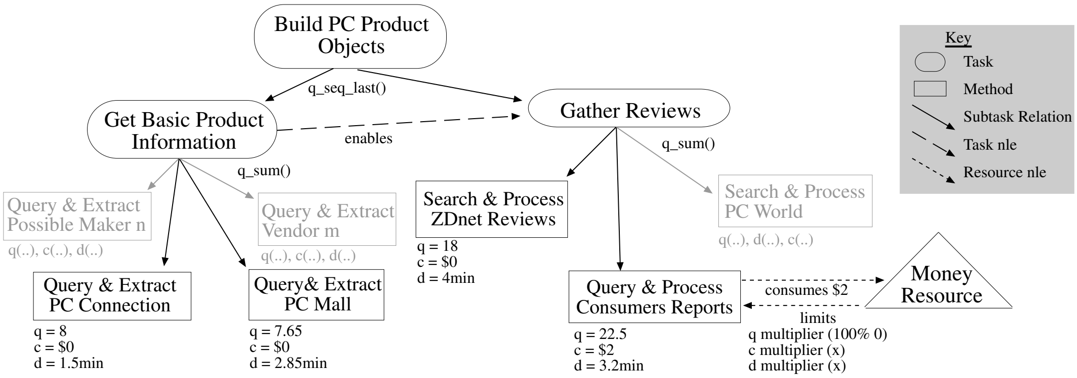


Figure 1: Simplified Subset of an Information Gathering Task Structure

The Design-to-Criteria scheduling problem is framed in terms of a TÆMS [11, 38] task network, which imposes structure on the primitive actions and defines how they are related. The most notable features of TÆMS are its domain independence, the explicit modeling of alternative ways to perform tasks, the explicit and quantified modeling of interactions between tasks, and the characterization of primitive actions in terms of quality, cost, and duration. We describe TÆMS in greater detail in Section 2 and in Section 3, we discuss how scheduling a TÆMS network can be mapped into a finite-horizon Markov Decision Process. However, to ground further discussion consider the TÆMS task structure shown in Figure 1. The task structure is a conceptual, simplified sub-graph of a task structure emitted by the BIG [25] information gathering agent; it describes a portion of the information gathering process. The top-level task is to construct product models of retail PC systems. It has two subtasks, *Get-Basic* and *Gather-Reviews*, both of which are decomposed into primitive actions, called *methods*, that are described in terms of their expected quality, cost, and duration. The *enables* arc between *Get-Basic* and *Gather* is a non-local-effect (nle) or task interaction; it models the fact that the review gathering methods need the names of products in order to gather reviews for them. *Get-Basic* has two methods, joined under the *q_sum()* quality-accumulation-function (*qaf*), which defines how performing the subtasks relate to performing the parent task. In this case, either method or both may be employed to achieve *Get-Basic*. The same is true for *Gather-Reviews*. The qaf for *Build-PC-Product-Objects* is a *q_seq_last()* which indicates that the two subtasks must be performed, in order, and that the resultant quality of the last subtask is the quality of the parent task; thus there are nine alternative ways to achieve the top-level goal in this particular sub-structure.

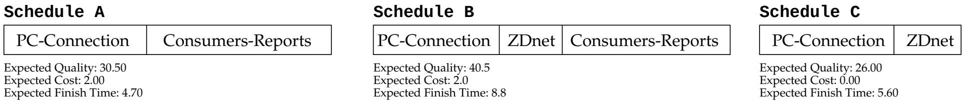


Figure 2: Different Schedules Produced for Different Design Criteria

Three different optimal schedules for achieving the top-level goal of the task structure, produced for three different sets of design criteria, are shown in Figure 2. Schedule A is constructed for a client who needs a high quality solution, requires the solution in seven minutes or less, and who is willing to pay for it. Schedule B is constructed to suit the needs of a client who has plenty of time and is willing to wait for a high quality solution, and is willing to pay for it. Schedule C is constructed for a client who has neither time nor money. Even this simple example illustrates the notion of quantified choice in TÆMS and how the Design-to-Criteria methodology leverages the quantification to build different schedules for different contexts. However, this example also illustrates a weakness in TÆMS as presented in Figure 1 – a weakness that is carried forward to the scheduling process and consequently to the schedules returned to the client. The initial design of TÆMS included only expected value modeling of primitive actions and task interactions. Subsequently, we have come to understand the strength of explicit modeling of uncertainty and the implications of these new models to the Design-to-Criteria scheduling process. Note that in this case, because the performance characteristics of TÆMS method executions are deterministic, the optimal policy produced by an MDP for the task network is also a simple linear schedule.

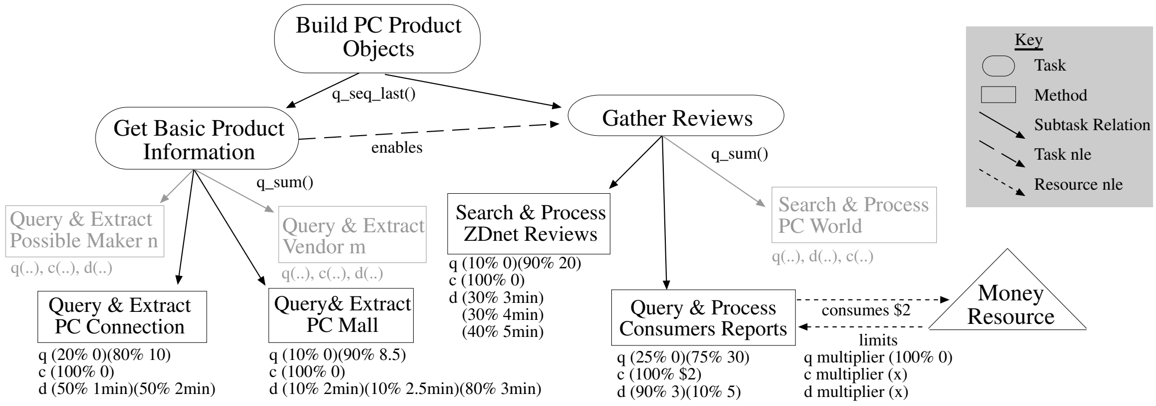
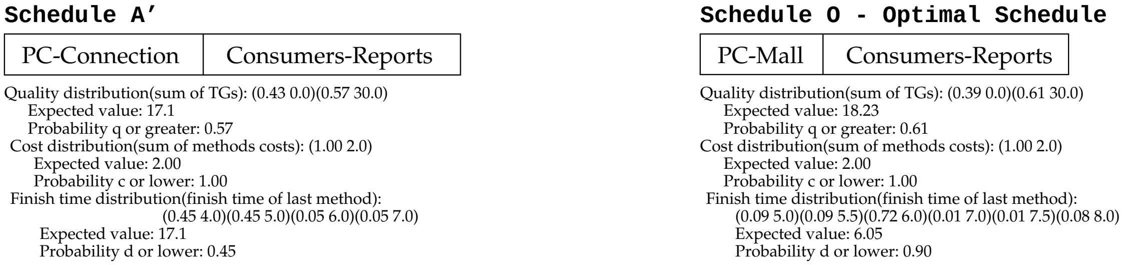


Figure 3: Simplified Subset of an Information Gathering Task Structure

Representing and reasoning about uncertainty of task execution behavior is one of the keys to scheduling computational structures when quality requirements and time and cost constraints are present. Additionally, with the inclusion of uncertainty modeling and propagation it is clear that there are many different dimensions and aspects of utility that can be used to evaluate the appropriateness of schedules. Consider the task of gathering information via the highly uncertain WWW to support a decision. Certain clients may prefer a risky information gathering plan that has a potentially high pay-off in terms of information gathered, but also has a high probability of failure. Other, more risk averse clients might prefer a course of action that results in a lower pay-off in exchange for more certainty about the pay-off and a lower probability of failure. Integrating notions of uncertainty in to the schedule evaluation process is one aspect of this work.

Prior to delving into an intellectual discussion of the role of uncertainty, consider the simplified task structure revised to include uncertainty, Figure 3, in the characterizations of the primitive actions. In the enhanced task structure, primitive actions are characterized statistically via discrete probability distributions rather than expected quality values. The quality distributions model the probability of obtaining different quality results and the possibility of failure (indicated by a zero quality result). Note that the expected values of these distributions are the same as those in the previous expected-value model, thus the structures are directly comparable. The cost and duration distributions represent the different possible costs and durations of the actions. This level of detail can be very important when reasoning about the gathering process. For example, in the enhanced model, it is clear that the method for querying and extracting text obtained from the *PC-Connection* site has a higher probability of failure than the method for querying and extracting text obtained from the *PC-Mall* site. In the original model, the detail is lacking and it is impossible to ascertain which method is more likely to fail.

The schedules shown in Figure 4 illustrate the value of uncertainty in this model from a scheduling perspective. Schedule A′ is identical to Schedule A from the expected value case (Figures 1 and 2), however, with the addition of uncertainty to the model, the scheduler can propagate uncertainty and create better estimates for the performance characteristics of the schedules. Note that the quality distribution for Schedule A′ includes a 20% chance of failure. In fact, with the addition of uncertainty to the model, analysis shows that Schedule A is no longer the optimal schedule for the client (who needs a result in 6 minutes or less and is willing to pay for it). Instead Schedule O (Figure 4) is the best choice. Even though the *PC-Connection* method has a higher expected value, the *PC-Mall* method has a lower probability of failure. Since a failure in one of these methods precludes the execution of *Query-Consumers-Reports* (via the task interaction), the issue of failure is not local to the methods but instead impacts the schedule as a whole. Thus, when uncertainty is modeled and propagated during the scheduling process, Schedule O is the optimal schedule as it has the highest net expected quality value and it still meets the client's deadline constraint.[1] In this case, Schedule O is also the optimal sequence of methods as determined by an optimal MDP – this is because even if *PC-Mall* fails there is no time to recover and perform an alternate method before the deadline. In other words, in this particular case, there is no added value to exploring all possible action sequences because of the hard deadline(Figure 7). We return to this discussion in Section 3.



Schedule A′

PC-Connection	Consumers-Reports

```
Quality distribution(sum of TGs): (0.43 0.0)(0.57 30.0)
    Expected value: 17.1
    Probability q or greater: 0.57
 Cost distribution(sum of methods costs): (1.00 2.0)
     Expected value: 2.00
     Probability c or lower: 1.00
 Finish time distribution(finish time of last method):
                          (0.45 4.0)(0.45 5.0)(0.05 6.0)(0.05 7.0)
      Expected value: 17.1
      Probability d or lower: 0.45
```

Schedule O - Optimal Schedule

PC-Mall	Consumers-Reports

```
Quality distribution(sum of TGs): (0.39 0.0)(0.61 30.0)
    Expected value: 18.23
    Probability q or greater: 0.61
Cost distribution(sum of methods costs): (1.00 2.0)
    Expected value: 2.00
    Probability c or lower: 1.00
 Finish time distribution(finish time of last method):
      (0.09 5.0)(0.09 5.5)(0.72 6.0)(0.01 7.0)(0.01 7.5)(0.08 8.0)
      Expected value: 6.05
      Probability d or lower: 0.90
```

Figure 4: Uncertainty Representation Changes optimal Schedule

This example conceptually illustrates one aspect of the value of uncertainty in the task models and in the scheduling process – better models lead to better schedules. Based on the observation that models containing uncertainty lead to more accurate representations and facilitate deeper analysis, the TÆMS task modeling framework was enhanced to model uncertainty about the quality, cost, and duration characteristics of tasks using discrete probability distributions. The modeling framework was also extended so that nles (task interactions) are also quantified and characterized using the describe probability distributions. We have augmented and extended the Design-to-Criteria scheduling system to leverage this new explicit representation of uncertainty to build better custom schedules. We have also constructed a secondary contingency-based schedule modification and selection algorithm that may be used in certain situations to ensure that recovery options exist if the chosen schedule fails. Both approaches can be thought of extending the space of options that the scheduler examines and also the analysis associated with determining the effectiveness of an option.

[1] Note that certain members of the *Consumers-Reports* method's duration distribution exceed the deadline of 6 minutes. In these cases, the method's results are considered unusable and zero quality is produced by the execution. Accordingly, for the density that exceeds the deadline the method's quality distribution is modified to reflect the additional probability of failure caused by exceeding the deadline. This is discussed in greater detail in Section 4.2.

This allows for performance guarantees on the choice of best schedule since it is more closely aligned to the sequence of actions suggested by the optimal control policy. Uncertainty plays several roles in the scheduling process:

Accuracy Uncertainty modeling enables the scheduler to represent and propagate uncertainty about tasks and their outcomes. This results in more accurate models of individual tasks, and more importantly, more accurate models of task sequences and task interactions. In contrast to reasoning from a single expected value, this enhancement supports notions like "30% of the time Task A will fail and 70% of the time it will generate high-quality results." Because the models of tasks, task interactions, and sequences of tasks are more accurate, the scheduler builds better schedules, as illustrated by Figures 3 and 4.

Focusing Uncertainty's second role is in *focusing;* the scheduler uses the client's design criteria throughout the scheduling process to focus efforts on building schedules and partial schedules that best satisfice, from a rational perspective [30, 31], to meet the criteria. This focusing behavior is what enables the scheduler to cope with the exponential combinators and produce results in soft real-time. When uncertainty reduction is important, the scheduler may select tasks that have a high degree of certainty about the specified dimension(s) and trade-off utility in other dimensions as specified by the client's criteria. For example, if certainty in the quality dimension is important to the client relative to raw quality goodness, the scheduler may trade-off high quality for more certainty about quality when building schedules, resulting in schedules with lower overall quality but higher quality certainty. In situations where a deadline must be met, the scheduler may elect to trade-off quality or even short duration, possibly in exchange for certainty about duration, producing schedules whose durations are not as short as possible, but whose durations are more certain than the schedules that have the shortest durations. These simple examples are members of a large class of multi-dimensional attribute trade-offs that Design-to-Criteria considers when building schedules [36, 37].

Construction The third use of uncertainty in the scheduling process is in *construction*; when uncertainty is important to the client, the scheduler may take a more active approach to uncertainty reduction and elect to use more than one way of achieving various tasks in order to increase the certainty of results in desired dimension(s).

Evaluation The fourth role of uncertainty is in *evaluation;* it enables the scheduler to evaluate quality, cost, duration, *and* uncertainty trade-offs when building custom schedules to meet a particular client's needs. The addition of uncertainty to both the task model and the goal/design criteria allows clients to specify how important, if at all, uncertainty reduction is relative to other schedule features like raw-goodness and threshold/limit specifications in each of the three modeled dimensions: quality, cost, and duration.

Contingency Analysis The fifth use of uncertainty is in the support of secondary *contingency analysis*. The general Design-to-Criteria scheduling process is designed to cope with exponential combinatorics and to produce results in soft real-time. However, its somewhat myopic approximation and localization methodologies do not consider the existence of recovery options or their value to the client. In the general case, explicit contingency analysis is not required. In the event of a failure, the scheduler is reinvoked and it plans a new course of action based on the current context (taking into consideration the successes as well as the failures, considering the value of results that been produced to the particular point). In hard deadline situations, however, the scheduler may not be able to recover and employ an alternative solution path because valuable time has been spent traversing a solution path that cannot lead to a final solution. Our uncertainty based contingency analysis tools can help in this situation by pre-evaluating the likelihood of recovery from a particular path and factoring that into the utility associated with a particular schedule. The improved estimates (based on the possibility of recovery options) can result in the selection of a different schedule, possibly one that leads to higher quality results with greater frequency. We return to contingency analysis in Section 5. Obviously, in the best of all possible worlds, instead of generating a schedule where we have to reschedule at failure points, we would like to construct an optimal meta-control policy which prescribes the next best action to take based on performance characteristics of the most recently executed primitive action. For most reasonable size task structures the computational overhead of constructing this policy online is unrealistic. However, we would like to see how well the contingency analysis approach performs in relation to optimal. In Section 3, we describe an approach for constructing such an optimal policy from our TÆMS task graph representation.

In general, the different implications of uncertainty to the scheduling process manifest themselves in two primary ways. One is with respect to the general scheduling process. By integrating and leveraging uncertainty within the framework of coping with combinatorics and generating custom schedules, we can produce better schedules in situations where certainty is important. Notions of redundancy, reducing uncertainty at schedule time, and focusing schedule generation on producing certain solutions are aspects of this facet. The other use of uncertainty is a detailed analysis that considers schedule recovery options and revises schedule expectations to reflect this more detailed analysis. On one hand there is the utilization of uncertainty in the approximate, satisficing, soft real-time computational Design-to-Criteria framework, and on the other hand there is an added expense, but a more thorough, detailed analysis that pays real dividends in hard-deadline situations that are accompanied by up front time for the extra analysis.

Design-to-Criteria [37, 36] traces its ancestry to the ideas of Design-to-Time [15, 16, 17] scheduling and to research in flexible computation [19] and anytime algorithms [7, 29, 41, 43]. Design-to-Criteria is related to Design-to-Time in that both scheduling methodologies are domain independent, operating on an abstract model of a particular problem solving process; more importantly both methodologies entail selecting from alternative ways to perform tasks, where each way has different performance characteristics, in order to construct custom schedules for a particular situation. Design-to-Time focused on quality and time trade-offs and building schedules to meet particular deadlines. To increase flexibility, Design-to-Criteria instead builds schedules that trade-off quality, cost, duration, and certainty in each of these dimensions, to meet a particular set of design criteria, in addition to meeting deadlines and other hard resource constraints. In the spirit of flexible computation, Design-to-Criteria also uses this trade-off analysis to control the scheduling combinatorics throughout the scheduling process, rather than as a post-production schedule selection mechanism as in Design-to-Time. [2]

This work falls into the general area of flexible computation [19], but differs from most flexible computation approaches in its use of multiple methods for task achievement (one exception is [20]), in its first class treatment of uncertainty, and in its ability to use uncertainty information in the selection of methods for execution. Much work in flexible computation makes use of anytime algorithms [7, 29, 41], algorithms that always have an answer at hand and produce higher quality results as they are given more time, up to a threshold. The TÆMS multiple methods approach can model any activity, including anytime algorithms[3], that can be characterized statistically and we place no constraints on the statistical behavior of the activities in question. In our work, uncertainty is a first class concept that both appears in the statistical descriptions of the available methods and is propagated and related as schedules and schedule approximations are generated. Unlike most work in anytime algorithms that focuses on the propagation of uncertainty [42], we can also include uncertainty and uncertainty reduction in the goal criteria and focus work on reducing uncertainty when important to the client. This ability stems from our task model's representation of alternative ways to perform various tasks. Because multiple-methods often exist to perform tasks, we can reason about the quality, cost, duration, and uncertainty trade-offs of different actions when determining which actions to perform, achieving the best possible overall results.

Recent research has advanced Design-to-Criteria in three primary areas: refining the goal directed criteria mechanism and trade-off analysis process, improving the quality estimates associated with final schedules, and the addition and incorporation of uncertainty in the scheduling process. In this paper, we focus on the uncertainty aspect of our recent work, though we point out other advances along the way. The approximate, trade-off behavior of the scheduling algorithm is presented in [37], along with identification of sources of complexity that pose significant obstacles to generating real-time schedules and doing so in soft real-time.

This paper is structured as follows. In Section 2 we discuss the TÆMS task modeling framework and the addition of uncertainty to the task models. In Section 3 we describe how the TÆMS task network is mapped to a finite-horizon Markov Decision Process and the optimal policy is computed. Section 4 discusses how uncertainty is integrated and leveraged in the main Design-to-Criteria scheduling process. In Section 5 we step outside of the main scheduling process and discuss secondary contingency analysis methodology that uses Design-to-Criteria to explore uncertainty and the ramifications of schedule failure. Experimental results illustrating the strength of contingency analysis, relative to Design-to-Criteria's myopic view, for certain classes of task structures are provided in Section 6.

[2] In Design-to-Time, schedule production is designed to produce an assortment of schedules, via a fixed set of heuristics, regardless of the design criteria. In Design-to-Criteria, where possible, all computation is directed at producing schedules, partial schedules, and schedule approximations that meet the design criteria, thus resulting in a larger set of high quality schedules from which to choose the "best" schedule to return to the client.

[3] Though if all actions were anytime algorithms, there are better ways to frame and perform the scheduling task.

2 Extending the TÆMS Modeling Language

TÆMS (Task Analysis, Environment Modeling, and Simulation) is a domain independent task modeling framework used to describe and reason about complex problem solving processes. TÆMS models are used in multi-agent coordination research [11, 38] and are being used in many other research projects, including: Cooperative-Information-Gathering [27, 25], collaborative distributed design [12], distributed situation assessment [6], surviveable systems [35], multi-agent diagnoses [3], intelligent environments [24], hospital patient scheduling [10], and coordination of software process [22]. Typically a problem solver represents domain problem solving actions in TÆMS, possibly at some level of abstraction, and then passes the TÆMS models on to agent control problem solvers like the multi-agent coordination modules or the Design-to-Criteria scheduler.[4]

TÆMS models are hierarchical abstractions of problem solving processes that describe alternative ways of accomplishing a desired goal; they represent major tasks and major decision points, interactions between tasks, and resource constraints but they do not describe the intimate details of each primitive action. All primitive actions in TÆMS, called *methods*, are statistically characterized in three dimensions: quality, cost and duration. Quality is a deliberately abstract domain-independent concept that describes the contribution of a particular action to overall problem solving. Thus, different applications have different notions of what corresponds to model quality. Duration describes the amount of time that the action modeled by the method will take to execute and cost describes the financial or opportunity cost inherent in performing the action. With the recent addition of uncertainty modeling, the statistical characteristics of the three dimensions are described via discrete probability distributions associated with each method. The uncertainty representation is also applied to task interactions like enablement, facilitation and hindering effects.[5] Thus agents may not only reason about the certainty of actions, e.g., "method A will fail 10% of the time," but also with respect to the interactions, e.g., "10% of the time facilitation will increase the quality by 5% and 90% of the time it will increase the quality by 8%," and the joint of these two. (Since interaction effects are dependent on the quality of the originator of the effect.) The quantification of methods and interactions in TÆMS is not regarded as a perfect science. Task structure programmers or problem solver generators *estimate* the performance characteristics of primitive actions. These estimates can be refined over time through learning [21] and reasoners typically replan and reschedule when unexpected events occur.

To ground further discussion, consider Figure 5, which is a slightly more complete view of the information gathering task structure introduced in Figure 1. The top-level task in this structure is *Recommend-a-High-End-PC-System* and it has two subtasks: one that pertains to finding information about products and constructing models of them, *Build-Product-Objects*, and one for making the decision about which product to purchase, *Make-Decision*. The two tasks are governed by a *q_seq_last()* qaf. Qafs specify how the quality of the subtasks is related at the parent task. With recent extensions to TÆMS, qafs may also specify orderings among the subtasks. Let T denote a task, c_i denote one of its children, and let n denote the number of children of T. Let q denote the quality of the item in question, e.g., T_q is the quality of the task and c_{i_q} is the quality of the ith child of T. In TÆMS, the quality of any task or method before performance (or after failure) is zero. A sampling of the qafs defined in TÆMS includes:

- *q_sum*: $T_q = \sum_{i=1}^{n} c_{i_q}$ and any of the subtasks may be performed (power-set minus empty-set) in any order.
- *q_sum_all*: $T_q = \sum_{i=1}^{n} c_{i_q}$ and all subtasks are to be performed in any order.
- *q_min*: $T_q = min(c_{0_q}, c_{1_q}, .., c_{n_q})$ and all subtasks are to be performed in any order. Since all tasks have zero initial quality, failure to perform a given child under a *q_min()* results in zero quality for the parent task.
- *q_max*: $T_q = max(c_{0_q}, c_{1_q}, .., c_{n_q})$ and any number of subtasks may be performed in any order, though generally only one task is selected.
- *q_exactly_one*: $T_q = EXOR\ (c_{0_q},\ c_{1_q},\ c_{n_q})$ and only one of the subtasks may be performed.
- *q_seq*: $T_q = c_{n_q}$ and all subtasks must be performed in order.

[4] In the process work, a translator transforms and abstracts process programs into TÆMS task structures for scheduling and coordination.

[5] Facilitation and hindering task interactions model soft relationships in which a result produced by some task may be beneficial or harmful to another task. In the case of facilitation, the existence of the result, and the activation of the non-local effect generally increases the quality of the recipient task or reduces its cost or duration.

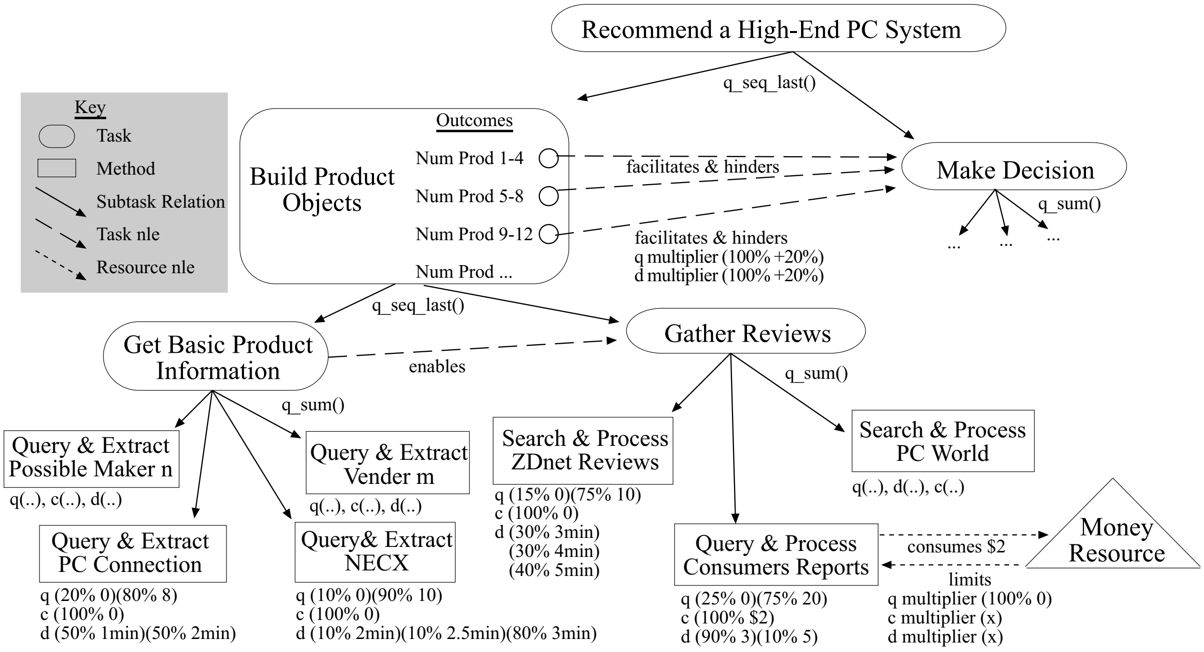


Figure 5: Information Gathering Task Structure

- *q_seq_sum, q_seq_min, q_seq_max, q_seq_last*: The *seq* prefix in this case denotes sequence preference and that all subtasks must be performed; the suffix denotes the function to perform with the resultant qualities, e.g., *q_seq_sum* indicates $T_q = \sum_{i=1}^{n} c_{i_q}$

Recommend-a-High-End-PC is thus performed by performing each of its subtasks, in order, and its quality is determined by the *Make-Decision* subtask. This models the fact that the decision process takes into consideration the quality, coverage, and certainty of the information used to make the decision and reflects these attributes in the quality of its output. As discussed, *Build-Product-Objects* is performed by performing each of its child tasks, in order, and its quality is the sum of its children's qualities. In contrast, *Get-Basic* and *Gather-Reviews* can be achieved by performing any one or more of their respective child tasks. Note the *enables* interaction between *Get-Basic* and *Gather-Reviews.* This non-local effect models a hard precedence relationship between the tasks – the agent must first successfully learn about products before it can locate reviews for them. In TÆMS, task interactions are triggered by conditions in the originator and the effects of the interactions are reflected in the quality, cost, and duration distributions of the recipient. With the addition of uncertainty to TÆMS, soft interaction effects like facilitation and hindering, are also quantified via probability distributions. Task interactions in TÆMS include: *facilitates*, *hinders*, *bounded facilitates*, *sigmoid*, *enables*, and *disables*.

Resource models are another recent addition to the TÆMS framework. The information gathering task structure also shows the use of a monetary resource. Resources are currently either consumable or non-consumable (replaced after use, e.g., a network), though the hierarchical resource models will support further specialization. Task resource consumption and production behaviors are modeled in TÆMS via *consumes* and *produces* task/resource non-local effects – these non-local effects describe the quantity of resources consumed or produced by task execution. In the event that resources are insufficient to meet the requirements of a given task, the negative effects are modeled via a *limits* resource-to-task non-local effect that is akin to a *hinders* task-to-task non-local effect, i.e., it expresses negative multiplier effects on the recipient's quality, cost, and duration distributions. For a non-consumable resource, e.g., network bandwidth, where the resource is diminished during the usage and then returned to its initial state, the definitions for consumes and limits are:

- A resource-centered non local effect is a function of the form: *nle(M, R, t, q, c, d, $R_{quantity}$, p1, p2,...): [method* × *resource*

× current time × method quality × method cost × method duration × resource quantity × parameter1 × other parameter2 ..] = [method quality × method cost × method duration × resource quantity]

$$consumes(M, R, t, q, c, d, R_{quantity}, \alpha_{quantity}, M_{t\_exec}) =$$
$$\begin{cases} [q, c, d] & and \\ R_{quantity} = R_{quantity} - \alpha_{quantity} & t > M_{t\_exec} \\ R_{quantity} & otherwise \end{cases}$$

$$limits(M, R, t, q, c, d, R_{quantity}, \alpha_{quantity}, M_{t\_exec}, \phi_q, \phi_c, \phi_d) =$$
$$\begin{cases} [q - q * \phi_q, c + c * \phi_c, d + d * \phi_d] & t > M_{t\_exec} \;\&\; R_{quan} < \alpha_{quan} \\ [q, c, d] & otherwise \\ R_{quantity} & \end{cases}$$

Another recent addition to TÆMS is the *outcome* construct. Outcomes model situations in which a given method has different classes of possible results, each class having its own distinct quality, cost, and duration characteristics and possibly even its own interactions with other tasks. The *Build-Product-Objects* task in Figure 5 illustrates the outcomes construct[6]; the outcomes serve to indicate the number of objects generated during the information gathering phase. Attached to each of these outcomes are hindering and facilitation soft non-local effects that affect the quality, cost, and duration of the decision making task. This models the notion that the time required to make the decision increases as more products are compared, but, that the decision process benefits in terms of quality by having more products.

TÆMS also supports modeling of tasks that arrive at particular points in time, individual deadlines on tasks, earliest start times for tasks, and non-local tasks (those belonging to other agents). Obviously, scheduling TÆMS task structures is a non-trivial process. In the development of TÆMS there has been a constant tension between representational power and the combinatorics inherent in working with the structure. The result is a model that is non-trivial to process and schedule in any optimal sense, but also one that lends itself to flexible and approximate processing strategies.

3 An Optimal Meta-Controller for TÆMS Task Network Scheduling

As discussed in the introduction, our heuristic approach to scheduling the TÆMS task network is to find a linear sequence of actions whose performance, as a result of limited search, best matches the goal criteria. We reschedule when the results of the partial execution of the action sequence indicates there is a high probability that we will no longer meet goal criteria. This is in contrast to an optimal meta-control policy which would specify the best possible method to execute at each instant so as to achieve the desired high-level goal while optimizing the criteria and resource specifications. The drawback with such an approach is that for most reasonable size task structures the computational overhead of constructing this policy online is infeasible. Additionally, our research concentrates on open, dynamic environments where the task set of the agent, or the agent's objectives, may change and the models used to produce the MDP are approximate and uncertain and of themselves. In this context, it is likely that the MDP would have to be frequently recreated and thus the intractability of the process becomes even more of an issue. However, we would like to see how well our heuristic approach performs in relation to the optimal policy in situations where it is computationally feasible to compute the policy off-line.

We define the TÆMS task network scheduling problem as a finite-horizon Markov Decision Process(MDP) which tries to maximize its expected accumulated reward given the criteria (quality, cost, duration) specification. It is a finite-horizon MDP because a primitive action can be executed only once in a particular execution path and hence there are no loops. The mapping of a TÆMS task structure to an MDP is fairly straightforward since TÆMS can be thought of as a compact representation of a class of MDP problems. TÆMS differs in that it implicitly describes the enumerated search space that is explicitly described by the MDP. From another view, the MDP unwinds TÆMS

[6] The actual information gathering task structure does not incorporate outcomes at the task level. This example is a conceptual abstraction of the class of task structures produced by the agent's planner and is simplified for example purposes. Outcomes at the task level have semantics that are difficult to specify.

into a state space – TÆMS does not represent the actual effects of the individual alternative paths nor does it break individual elements from quality, cost, or duration distributions. TÆMS is a specification of tasks, their relationships, their interactions, and their statistical characteristics. It does not carry through the implications of choices. The MDP framework, on the other hand, explicitly describes each possible individual execution characteristic of each primitive action. Additionally, TÆMS specifies constraints on an ordering rather than explicitly representing the implications of the ordering. There is no requirement of immediate precedence and no constraint on immediate succession either. An MDP representation would lay out exact precedence and succession orderings of methods within a path in the MDP tree. Additionally, the compact process-style representation of TAEMS is a more natural representation of agent activity.

The translation process of a TÆMS task structure to a MDP involves following a procedure which lays out each possible execution path for achieving the high level goal. The MDP translation is a procedure which allows for the transformation of a TÆMS task structure *T* to the corresponding MDP *M*. The state in the MDP representation is a vector which represents the methods that have been executed in order to reach that state along with their execution characteristics. The MDP action is the execution of a particular method. MDP actions have outcomes and each outcome is characterized by a 3-tuple consisting of discrete quality, cost and duration values obtained from the expected performance distribution of the MDP action. The rewards are computed only for the terminal states, i.e. the intermediate states have null rewards. The reward is computed by applying a complex criteria evaluation function of the quality, cost and duration values obtained by the terminal state which is described in Section 4. Value iteration is the dynamic programming algorithm used to compute the optimal policy. In theory, value iteration requires an infinite number of iterations to converge to the optimal policy. However, in practice, we stop once the value function changes by only a small epsilon threshold in a sweep. The following is the algorithm for the translation process.

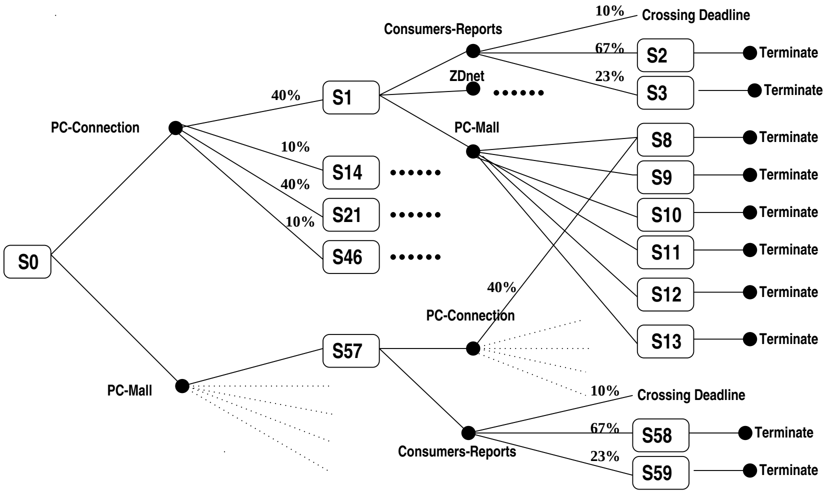


Figure 6: Translation process from TAEMS task structure to Markov Decision Process for Simplified Subset of an Information Gathering Task Structure.

Let TG be the top level goal in *T* and let METHODS be the set of primitive actions in *T*.

```
1. Initialize MDP with state s;
2. Translate(s)
   (a) Identify the set of actions(subset of METHODS) which are possible from s.
   (b) Iterate over each action
       i. If action is not TERMINATE
          A. Expand each outcome(characterized by discrete quality, cost, duration values).
          B. Determine if outcome can lead to a new state while adhering to the criteria constraints.
          C. if new state $s_{prime}$ is reached, Translate($s_{prime}$)
```

ii. Else if action is TERMINATE, set reward of terminating state to be a function of the quality, cost and duration values of the state.

3. ValueIteration(StateSet);

4. Return optimal policy.

To make this discussion on the translation process more concrete, we will apply a few iterations of the algorithm on the task structure described in Figure 3. Lets assume the client design criteria specifies that the task should achieve the maximum possible quality within a hard deadline of 6 minutes. Upon translation, the corresponding MDP has 76 states. Figure 6 describes the translation process in progress. The input to the algorithm is the task structure in Figure 3 described in textual format. The start state *SO* is initialized. The PossibleActionSet for state *SO* is *PC-Connection*, *PC-Mall*. *ZDnet* and *Consumers-Reports* are not valid actions since their parent task *Gather Reviews* has an inactive incoming enables from *Get Basic*. The outcomes of each action in the PossibleActionSet are processed starting with *Consumers-Reports* which has 4 outcomes resulting in the states *S1*, *S14*, *S21* and *S46* respectively. The PossibleActionSet for state *S1* consists of *Consumers-Reports,* ZDnet, PC-Mall. The action *Consumers-Reports* has 4 outcomes. Two of the outcomes result in the total duration crossing the deadline of 6 minutes and hence result in a null state. The other two outcomes result in states *S2* and *S3* with probabilities 0.67 and 0.23 respectively. The outcome resulting in state *S2* has a quality of 30.0, cost of $2 and duration of 5 minutes. The PossibleActionSet for state *S2* contains only the *Terminate* action since the outcomes of all other possible actions result in the deadline being crossed. The current loop is exited when a *Terminate* action is encountered and also if a deadline is crossed resulting in null states. The rest of the MDP tree is extended in the same depth-first fashion. In Figure 6, the tuple <state=*S1*, action=*PC-Mall*, outcome = o1> results in state *S8* and so does <state=*S57*, action=*PC-Connection*, outcome = o1>. This is because the two tuples are just permutations of the same set of action outcomes.

The optimal policy for the above problem is shown in Figure 7. As we can see the policy suggests the method sequence {*PC-Mall, Consumer-Reports*} as the best schedule. There are no possible recoveries in the event of failure of either method within the deadline. In this particular case, because there are no recovery options, the single pass view of DTC produced the optimal solution, as discussed in Section 1. This is not always the case and the utility of the MDP expansion approach is made clear by the secondary contingency analysis research, presented in Section 5, that more closely approximates the properties of the MDP.

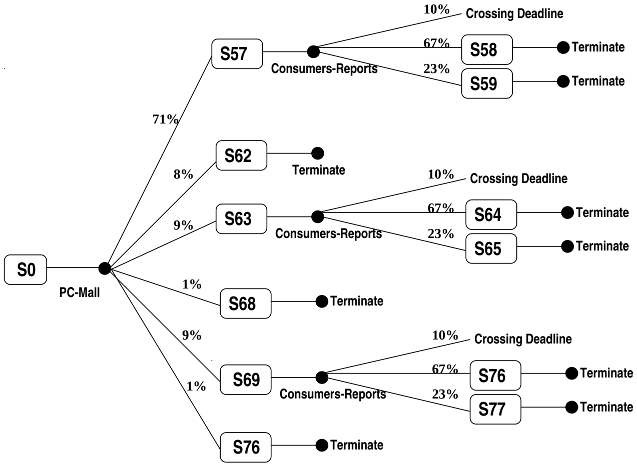


Figure 7: Optimal Policy for Simplified Subset of an Information Gathering Task Structure

As mentioned earlier, the MDP state space of task structures modeling real-world applications undergoes a combinatorial explosion. For instance, the number of states for a task structure with 4 methods with each method having 2 outcomes is in the 100's. However, a task structure with 6 methods with each having an average of 5 outcomes has about 30000 states. Hence, this approach has mainly been researched to serve as an off-line theoretical benchmark to evaluate our real-time heuristic schedulers.

4 Integrating Uncertainty Into Design-to-Criteria

Design-to-Criteria is the process of coping with exponential combinatorics to produce schedules in soft real-time that meet a particular set of design criteria and hard constraints like deadlines or cost limitations. This scheduling problem requires a sophisticated heuristic approach because of the task's inherent computational complexity. To understand the complexity and get a feel for the scheduling process, consider a task structure only a single level deep, where a single task has n children that are methods and it accumulates quality according to the *q_sum()* qaf. In this case, there are $2^n - 1$ unordered sets of methods that can be used to achieve the parent task, and within each set of m methods, $m!$ possible orderings of methods in the schedule. Conceptually, the number of unordered sets is due to planning style complexity where the issue is deciding which methods to use to bring about a desired state. The number of sequences of each unordered method set is the classic scheduling side complexity where ordering is the focus. In general, the upper-bound on the number of possible schedules for a TÆMS task structure containing n methods is given in Equation 1.

$$\sum_{i=0}^{n} \binom{n}{i} i! \tag{1}$$

The combinatorics are pronounced ($\omega(2^n)$ and $o(n^n)$ by Stirling's approximation) and in practice the generation of a provably optimal solution through generation of all possible schedules is infeasible. In our research, we often schedule task structures having between 25 and 180 methods, e.g., [25, 34]. A sample task structure with 25 methods produces over 67 million different unordered method sets and over 1.1×10^{25} possible schedules. The production of provably optimal solutions through more refined techniques, such as dynamic programming or a state-based-search, has been unsuccessful due to the number and types of constraints present in TÆMS, e.g., interactions between tasks, start times, deadlines, commitments associated with tasks, unruly quality, cost, and duration characteristics, etc. In DTC, the scheduler controls the combinatorics and produces schedules through a satisficing methodology described in detail in [37] and [34, 36, 14, 15].

This practical limitation also applies to the production of an optimal control policy via MDP – in this case, the combinatorics are even worse because the MDP "unwinds" the TÆMS representation and for each triple of $<$ quality, cost, duration $>$ in a method's outcome, the MDP contains one state (Section 3). With respect to optimal policies versus optimal schedules, the real question is: *In what situations does the sequence of actions in the best schedule differ from the sequence of actions in the optimal policy?* Another way of understanding the problem is how close can we approximate the performance of an optimal policy through the process of incremental rescheduling, and how often does this rescheduling need to occur? Note that the scheduling view and the policy view differ. The policy view explores all possible method sequences, including recovery from method failure; the scheduling process composes and end-to-end view of the agent's process while propagating and reasoning about uncertainty but not exploring recovery options. In other words, a schedule is constructed with the assumption that rescheduling will occur with failure or when the results deviate from those expected by the schedule In contrast, a policy defines which method to execute next based on the outcomes of previous method executions. With scheduling and rescheduling we hope to approximate the policy, however, even if rescheduling occurs after each method execution the performance of the scheduler may not be identical to that of the optimal policy because with each decision, each choice, the scheduler does not consider all possible method sequences from the current state. The policy view is stronger, but, also infeasible in our research due to the size of the problem space and the dynamics involved. DTC is deployed in multi-agent environments where the models of the environment and the agents' processes are imperfect and new tasks or new requests may arrive at any point in time. The combinatorics combined with the dynamism have placed hard constraints on the scheduling problem. In this paper we do not revisit the satisficing methodology of DTC in detail, but instead discuss the implications of uncertainty to task modeling and the scheduling process, and introduce an extension to DTC to support contingency analysis and explicit planning for recovery that more closely approximates the performance and behavior of optimal MDP produced policies (Section 6).

The main facets of DTC's methodology include:

Goal Directed Focusing The client's design criteria is 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.

This is achieved by creating and identifying partial solutions that seem likely to meet the criteria and concentrating further development on these classes of partial solutions, pruning or ignoring other partial solutions that are deemed least probable to lead to "good" solutions.

Approximation Schedule approximations, called *alternatives*, are used to provide an inexpensive, but coarse, overview of the schedule solution space. Alternatives contain a set of unordered actions that can be scheduled (ordered) to achieve a particular task along with estimates for the quality, cost, and duration distributions that may result from scheduling the actions. An alternative represents one *possible* way in which a given task may be achieved. Alternatives are inexpensive to compute as the complex task interactions are only partially considered and ordering, resource, and other constraints are ignored. The alternative abstraction space is used in conjunction with criteria directed focusing to build schedules from alternatives that are most likely to lead to good schedules.

Heuristic Decision Making The process of turning alternatives into schedules, i.e., sequencing a set of actions, is the classic scheduling problem and this too suffers from high order complexity; $O(n!)$ to schedule a set of n unordered actions. We cope with this complexity using a group of heuristics for action ordering. The heuristics take into consideration task interactions, attempting to take advantage of positive interactions while avoiding negative interactions. They also consider resource limits, individual action deadlines, task deadlines, commitments made with other problem solving agents, and other constraints. The heuristic algorithm reduces the $O(n!)$ action ordering problem to low-order polynomial levels in the worst case.

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 the high-level task but do not live up to quality, cost, duration, or certainty expectations set by the estimates contained in the alternatives. This can be caused by an over constrained problem, but also by complex task interactions that are glossed over by the alternative approximation and not considered by the action ordering heuristics. A secondary set of improvement [44, 32] heuristics act as a safety net to catch the errors that are correctable.

The addition of uncertainty modeling to TÆMS has several implications to the Design-to-Criteria scheduling process. First, the client must be provided a mechanism to describe the relative importance of certainty or uncertainty reduction to their application. In some situations, certainty may not be an issue, but in other situations certainty may be highly important. An example of this in a multi-agent context is the certainty of an agent to fulfil its commitment to provide a result to another agent before a specified time. Second, given the ability to specify certainty preferences, how can the information be used in the scheduling process to produce schedules that are more or less certain, i.e., how to design schedules to address the enhanced design criteria. Third, is the issue of how the new uncertainty representation impacts the computations and analysis performed by the scheduler – the questions are whether or not existing computations are affected by the new model and whether or not the computations can be improved. Related to this is the issue of building models of schedules where the schedule characteristics include uncertainty and the relationship of a distribution style representation to a single value representation like a hard deadline or hard cost constraint.

In the following sections we describe how these issues are addressed in Design-to-Criteria. In Section 4.1 we discuss the integration of uncertainty into the client design criteria and how this is mapped to utility that is used during the scheduling process. Section 4.2 discusses how uncertainty, and the design criteria, are used in the scheduling process to produce more certain schedules when uncertainty reduction is important to the client. Section 4.3 identifies areas in which the computations are effected by the addition of uncertainty and how the representation of uncertainty is used in the modeling and construction of schedules. A high-level example of uncertainty reduction in the scheduling process is then given in Section 4.4. In a certain sense, integration of uncertainty in the main scheduler is done on a schedule by schedule basis, in Section 5 we step outside of the main scheduling process and discuss a secondary analysis process that goes beyond the independent view of schedules and instead considers recovery or *contingency* options for schedules.

4.1 Uncertainty in Client Goal Criteria and its Mapping to Utility

The client goal or design criteria is generated using a specification metaphor called *sliders*, the GUI shown in Figure 8. This metaphor was first presented in [36] and extended in [37, 34]. Sliders take on values from 0 to 100% and are arranged in slider banks where each bank contains a slider for quality, cost, and duration. The sum of the sliders in each bank range from 0 to 100%. A 100% weight given to a particular slider expresses that the slider in question is the only item of importance relative to the other sliders in the same bank. Examples follow below.

Intellectually, the research issues being addressed by the slider metaphor are not whether the metaphor is appropriate for all applications, but, rather the recognition that agents have different objectives in different situations and that when problem solving or scheduling with these pronounced combinatorics, control must be focused by the current objectives of the agent. The point is flexibility in control and the ability to dynamically focus the entire scheduling process on reasoning about the objectives (via design criteria), rather than using implicit, hardwired objectives in a *post-hoc* fashion as in the previous Design-to-Time scheduling work [14, 15]. If the objectives are not considered during generation, in general, a much larger portion of the solution space must be searched to produce results useful to the agent. Casting Design-to-Criteria as a search process, this is akin to concentrating search on the classes of solutions and partial solutions that best meet the agent's objectives from start to finish, rather than as an evaluation specification applied to final states. In terms of an approximate MDP, this is conceptually analogous to only transitioning from states that correspond to some estimation of the agent's objectives as the other solution states are less desirable. The slider metaphor or goal specification could easily be replaced or extended – once control is designed to use the specification to *evaluate* the utility of a given solution or partial solution, changing the specification is straightforward. In this paper we do not concentrate per se on the constructionist roles of the goal specification, but instead describe the specification and how it relates to the evaluation of candidate options and extend the specification to include balancing uncertainty reduction with the other trade-offs evaluated by the scheduler. The use of the specification in evaluation is the central concern here as it determines which approaches to achieving a particular task, or the root level task, are more valuable than others. The constructionist aspects of the scheduling process, presented in [37, 34], dovetail to resolve task constraints once the choice of options has been made.

Regardless of the simplicity of changing the metaphor, the metaphor presented here has been proven to be appropriate for TÆMS and our applications because, in general, it is difficult to estimate *a priori* what is possible for a given task structure. For example, for a TÆMS task structure having as few as eight methods there are 109,600 possible schedules. Often we deal with task structures having between 25 and 180 methods. Scheduler clients (humans, planners, other control problem solvers like coordination modules) simply do not know whether a given hard deadline or hard preference may be met or even have an approximate idea of the types of solutions that are possible. Often they must describe relative preferences or relative trade-offs and then evaluate the class of solutions produced by the scheduler and possibly refine their overall objectives or their goal criteria accordingly. This is why some of the basic principles of the metaphor include satisficing and relative evaluation as well as client specified limits and thresholds on the different attributes. Extensions to support multiple criteria sets and preferences between solution classes are presented in [34].

There are five banks in the current specification metaphor, each relating to a different class of concerns:

Raw Goodness This bank describes the raw relative importance of each dimension. For example, setting the quality slider to 50% and cost and duration to 25% expresses the notion that high quality is twice as important as low cost and low duration. The label "raw" here denotes that this preference is not with respect to any particular deadline or other constraint. Because of the combinatorics of the scheduling problem, clients often do not know *a priori* what is possible for a given task structure and thus setting hard limits and thresholds can be problematic. This bank enables clients to specify simple, relative preferences about quality, cost, and duration.

Threshold and Limits This bank allows the client to set *soft* limits and thresholds for quality, cost, and duration either using a fixed limit/threshold value or using a utility function that describes gradual changes in utility as the value increases beyond a certain limit or as it crosses a certain threshold. The preferences expressed here are soft in that the scheduler may elect to cross a particular limit or threshold *if* the overall utility of the item in question is higher than the other candidates that stay within the limit or threshold. This concept is made more clear below when we describe how the design criteria is related to the utility used by the scheduler. It is important to note that hard constraints, e.g., hard deadlines, do exist in the scheduling process, but that the general design criteria is about the expression of relax-able constraints and soft general preferences.

Certainty Whereas the raw goodness set above expresses the relative importance of quality, cost, and duration, this set expresses the relative importance of uncertainty about quality, uncertainty about cost, and uncertainty about duration. For example, if a client does not actually care when a result is produced, but is going to schedule a meeting to discuss the results as soon as they are produced, the client would specify a preference for high certainty in the duration dimension, expressed as a significant weight given to the duration slider in this bank, e.g., 80% or 100%. This bank expresses relative predictability preferences.

Certainty Thresholds Like the thresholds and limits bank, this bank expresses the relative importance of meeting certainty thresholds in the quality, cost, and duration dimensions. For example, through this mechanism, clients can express a preference for schedules that have a duration certainty of 75% or higher (meaning that 75% of the time, the schedules will achieve their predicted runtime). As with limits and thresholds on quality, cost, and duration, it is typically difficult for clients to know *a priori* what certainty thresholds are possible for a given task structure so this bank expresses soft or relax-able preferences.

Meta This slider set relates the importance of the four previous slider sets. This separation allows clients to focus on relating quality, cost and duration with each other in each of the cases above, then to “step back” and decide how important each of the different classes are relative to each other. For example, within the raw goodness bank, client’s can reason about the relative importance of quality, cost, and duration, then do the same in the certainty bank, then decide how raw goodness relates to certainty. If certainty is the primary issue, then it is given more weight in the meta bank than raw goodness.

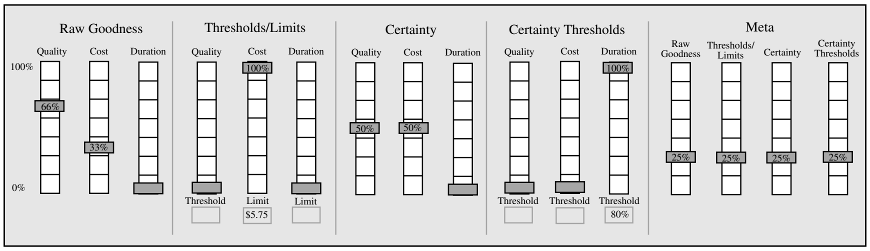


Figure 8: Goal Specification Metaphor

The incorporation of uncertainty into the criteria specification provides clients with a means to describe how important reducing uncertainty is for their application *relative* to raw-goodness and limits/thresholds. Given the ability to specify the importance of these attributes, the issue then becomes how to relate the attributes to utility that can be used in the scheduling process to evaluate and select from different possible courses of action. The mapping from sliders to utility is presented in [36], however, we must examine a portion of the computations in order to discuss the use of uncertainty in the utility computation as well. In general, utility is computed by comparing the statistical characteristics of a member of a set of candidate schedules to the observed characteristics for the set as a whole. The utility computations form the basis of the goal or design directed problem solving behaviors of the scheduler and are used both on completed schedules and the aforementioned alternatives (schedule approximations).

The utility computation is based on notions of *relative goodness* and *normalized comparison*. The computation is decomposed into components, with one component associated with each slider bank. The components are further decomposed into subcomponents, with one subcomponent associated with each slider in a particular bank, i.e., there is one subcomponent for quality, one for cost, and one for duration, in each bank. The subcomponents are summed to produce the rating component for a particular bank. Subcomponents are computed by examining items being rated in the particular dimension with which the subcomponent is associated. For example, to compute the component for the raw goodness (the first) slider bank:

1. Find the min and max expected values for quality, cost, and duration that occur in the set of schedules or alternatives being rated.
2. Loop over the set of alternatives or schedules to be rated and calculate the raw goodness rating for each by calculating the quality, cost, and duration subcomponents as follows in Steps 3, 4 and 4.
3. Let eq denote the expected quality value of the alternative or schedule under consideration.[7] Its quality subcomponent is a function of the percentage of quality achieved by eq relative to the min and max, min_q and max_q, quality values of the set of items being rated, scaled by the raw goodness quality slider, RG_slider_q and the total number of points in the raw goodness bank.

$$rating_q = \frac{(eq - min_q)}{max_q - min_q} * \frac{RG\_slider_q}{\sum_{i=q}^{d,c} RG\_slider_i}$$

4. Duration is different than quality as greater duration is generally less preferable. Whereas with the quality related equation, achieving the best quality of all items in the set should bring the highest reward, in this case, achieving the least duration of all items in the set should bring the highest reward. Cost is like duration in that lower cost is better.

$$rating_d = \frac{(max_d - ed)}{max_d - min_d} * \frac{RG\_slider_d}{\sum_{i=q}^{d,c} RG\_slider_i}$$

$$rating_c = \frac{(max_c - ec)}{max_c - min_c} * \frac{RG\_slider_c}{\sum_{i=q}^{d,c} RG\_slider_i}$$

5. The quality, duration, and cost subcomponents are then summed to obtain the aggregate raw goodness rating component.

The certainty rating subcomponents are computed like the previous subcomponent in that utility is computed by comparing a given item to the observed minima and maxima for the set of candidate items. However, the subcomponents differ in that the focus is on the certainty associated with the expected values of the quality, cost, and duration dimensions rather than the expected values themselves. Consider the quality case. The general idea is to reward schedules or alternatives based on how likely it is that a quality value that meets or exceeds the expected value will actually occur.[8] The reason for this is semantic – more quality is always a good thing. Clients will not mind if the resulting quality is greater than predicted, only if the resulting quality is less than predicted. Certainty about cost and duration is computed similarly, albeit that what is "good" is reversed – less cost and less duration are good things, thus, the probability of producing a result in less time or lower cost is combined with the probability of obtaining the expected (predicted) cost or duration.

Thus we compute the probability that the quality, as expressed by the discrete probability distribution, is greater than or equal to the expected value, we then normalize and scale the probability as with the previous components, and finally multiply by the proportion of points allocated to the certainty quality slider. Consider a partial example, if a schedule has a simple quality distribution that denotes 25% of the time 0 quality will result and 75% of the time quality 10 will result, its resulting expected quality value is 7.5. Contrast this with a schedule whose quality distribution denotes that 50% of the time 0 quality will result and 50% of the time 15 quality will result; its expected quality is also 7.5. However, the probability that the first schedule will generate a quality value greater than or equal to the expected value is .75 whereas the second schedule's probability is only .50. This is the gist of the certainty rating subcomponents – the more certain that the expected value, or a better value, will occur, the greater the reward. The calculation procedure is similar to the raw quality goodness procedure presented above, though the focus is always on probabilities and probabilities of the items being rated are normalized using the derived min and max probabilities for

[7] Schedules, schedule approximations, and partial schedule approximations are evaluated using the design criteria. The term "alternative" is used to denote a schedule approximation or a partial schedule approximation.

[8] An alternate interpretation is to determine the probability that the actual value will fall near the expected value, on the upside or the downside.

the set. For example, the computation to compute the quality certainty rating subcomponent is shown in Equation 2. In the equation, q denotes the quality distribution of the item being rated (schedule or alternative), eq denotes the expected value of the quality distribution, $Prob(q >= eq)$ determines the probability that quality exceeds the expected value, $min\_probability_q$ denotes the lowest observed instance of $Prob(q >= eq)$ and $max\_probability_q$ denotes the greatest observed instance of $Prob(q >= eq)$. The other components scale the rating according to the weight given to the associated slider.

$$r_q = \frac{(Prob(q >= eq) - min\_probability_q)}{max\_probability_q - min\_probability_q} * \frac{Certainty\_slider_q}{\sum_{i=q}^{d,c} Certainty\_slider_i}$$

The certainty threshold rating component differs from the general certainty component in that the boundaries are not determined by examining the candidate set of items being considered, but are, instead, specified by the client. Exceeding a particular certainty threshold results in the same utility regardless of how far a particular item exceeds the threshold. The initial conceptualization of this computation included a relative scaling component, i.e., the farther the distance above the threshold, the more utility. However, this resulted in a mismatch between expectations and results as certain schedules would receive greater utility from both the raw certainty bank and the certainty thresholds bank. The computation is again computed by iterating over the set of candidate items and computing utility subcomponents for each of the dimensions; the subcomponents are then again summed to produce the certainty thresholds rating component. To illustrate the general certainty threshold computation, consider the quality subcomponent computation:

$$\begin{aligned} & if\ \ (Prob(q >= eq) \geq client\_specified\_threshold_q)\ then \\ & rating_q = \frac{Certainty\_threshold\_slider_q}{\sum_{i=q}^{d,c} Certainty\_threshold\_slider_i}\ else \\ & rating_q = 0 \end{aligned}$$

4.2 Incorporating Uncertainty in the Design-to-Criteria Process

Uncertainty is integrated into the process of schedule production in two primary ways. First, certainty preferences specified in the client goal criteria are mapped into utility values which are used during the scheduling process to focus production on schedules and schedule approximations that best address the client's goals. If the objective is to produce highly certain results, the scheduler will thus evaluate the different statistical trade-offs of different possible actions accordingly, perhaps producing highly certain schedules whose expected quality is somewhat lower than the maximum possible quality for the task structure. This property is in some sense automatic as integrating certainty preferences into the criteria changes the choices made by the scheduler throughout its process[9]; different choices of different options based evaluation that includes certainty trade-offs results in a different set of final schedules.[10] The second use of uncertainty in the main scheduling production process is more direct. Through the addition of uncertainty to the TÆMS modeling framework and the goal specification, the scheduler can do additional analysis during schedule production to explore a larger, different, schedule space. Namely, when uncertainty reduction is important to the client, the scheduler can consider redundant activities for task achievement and consider moving uncertain activities earlier in the schedule to leave more time for recovery.

In order to illustrate the first type of integration, that flowing from the goal and utility specification pair, it is necessary to describe certain aspects of the scheduling process. Unlike traditional scheduling tasks where the primary issue is how to order a particular set of methods, Design-to-Criteria must also consider the many possible combinations of alternative approaches for achieving the high-level task. Prior to the process of building schedules, the traditional method-ordering scheduling problem, the scheduler must enumerate the different ways that the high-level tasks can

[9] For example, choices made at interior nodes, choices made at the root node, choices pertaining to which schedule approximations or partial approximations to develop, etc.

[10] This also illustrates the strength of integrating a separate dynamic specification of an agent's objectives into the control problem solving process. As discussed, other attributes and other trade-offs could easily be incorporated into the criteria in a similar fashion.

be achieved. Each "way" is a cheap to compute schedule approximation called an *alternative*. Alternatives contain unordered sets of methods and estimates for the quality, cost, and duration distributions that would result from building a schedule from the alternative. Alternatives differ from schedules in that the ordering for the methods has not yet been defined and the attribute estimates are computed without regard for complex task interactions or individual task-centric constraints like hard deadlines. This approximation is necessary because in order to evaluate the individual constraints and interactions, all the other methods in the *potential* schedule must be evaluated. The problem is circular – to evaluate method x in one alternative may require the evaluation of methods y and z, that are not in said alternative, which may in turn require evaluation of h and i, also not in the alternative, and so forth. In essence, full evaluation of a given method drags in the worst-case exponential combinatorics of the general TÆMS scheduling problem, hence the reliance on an approximation that gives a feel for the partial solution space at the local node.

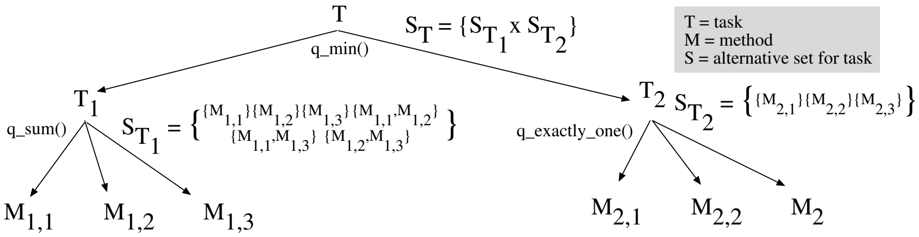

T
$S_T = \{S_{T_1} x S_{T_2}\}$
T = task
M = method
S = alternative set for task
q_min()
T_1
q_sum()
$S_{T_1} = \{ \{M_{1,1}\}\{M_{1,2}\}\{M_{1,3}\}\{M_{1,1},M_{1,2}\} \{M_{1,1},M_{1,3}\} \{M_{1,2},M_{1,3}\} \}$
T_2
$S_{T_2} = \{ \{M_{2,1}\}\{M_{2,2}\}\{M_{2,3}\} \}$
q_exactly_one()
$M_{1,1}$
$M_{1,2}$
$M_{1,3}$
$M_{2,1}$
$M_{2,2}$
M_2

Figure 9: Alternative Sets Lead to Cumbersome Combinatorics

Alternatives are constructed bottom-up from the leaves of the task hierarchy to the top-level task node, i.e., the alternatives of a task are combinations of the alternatives for its sub-tasks. Figure 9 shows the alternative set generation process for a small task structure. Alternatives are generated for the interior tasks T_1 and T_2 and then these alternatives are combined to produce the alternative set for the root task, T. The complexity of the alternative generation process is pronounced. A task structure with n methods leads to $O(2^n)$ possible alternatives at the root level. We control this combinatorial complexity by focusing alternative generation and propagation on alternatives that are most likely to result in schedules that meet the spirit of the client's goal criteria; alternatives that are less good at satisficing to meet the goal criteria are pruned from intermediate level alternative sets. For example, a criteria set denoting that certainty about quality is an important issue will result in the pruning of alternatives that have a relatively low degree of quality certainty.

After the alternative set for the high-level task is constructed, a subset of the alternatives are selected for scheduling. Again, complexity is the issue. For alternatives that have n methods, schedule construction via exhaustive search, $O(n!)$, is not feasible and even our low-order polynomial heuristic approach [37] precludes building schedules for all alternatives. Satisficing with respect to the client's goal criteria is used at this stage to select the alternatives that are most likely to lead to schedules that fit the criteria. As with alternative generation, if uncertainty is important to a particular client, schedules that reduce uncertainty in the desired dimensions will be produced. Using the heuristic approach, selected alternatives are scheduled by iterating over the set of unscheduled and unordered candidate methods and passing each method through a set of rating heuristics. The rating heuristics enforce hard constraints and express preference over the relaxation of soft constraints, e.g.:

- Enforce hard task interactions like enables and disables.
- Enforce hard resource constraints.
- Enforce earliest start times and deadlines.
- Try to take advantage of positive soft non-local effects, where doing one activity before another improves overall utility.
- Try to avoid negative soft non-local effects, where doing one activity before another degrades overall utility.
- Try to satisfy external commitments made with other agents[11] and avoid violating them (where commitments have varying degrees of importance).

[11] Applicable when Design-to-Criteria is used in a multi-agent context. In general, Design-to-Criteria interfaces with an external multi-agent coordination module, e.g., GPGP [10, 11], that proposes and forms commitments with other agents to handle the temporal sequencing of interdependent tasks.

- Try to coordinate[12] over soft-degradation style resource consumption and production.

Focusing is Design-to-Criteria's key to coping with the combinatorics and producing good schedules. This focusing methodology is analogous to generating only portions of the space of possible schedules – or akin to control using an approximate MDP (discussed further in Section 7). Figure 10 illustrates the scheduler's ability to focus processing on the goal criteria at hand. The figure shows the root-level alternative sets generated for two different criteria specifications; one where raw quality is the only factor of importance and one where certainty about quality is the only factor of importance. The task structure in question is moderately complex and has approximately 4×10^9 possible alternatives at the root level if focusing is not used to reduce the number of alternatives generated. When quality is the only factor, the alternatives generated have a high expected quality but also considerable quality uncertainty. In comparison, the alternatives generated for the quality certainty case have lower expected quality but a much higher degree of certainty. The distributions are statistically significantly different in both the quality and quality certainty dimensions; one-tailed t-tests reject the null hypothesis of equivalence at the .05 level. If a third case where quality and quality certainty are equally important (omitted for clarity), was added to the figure the alternatives would fall partly in the quality only range and partly in the certainty only range; the overlap is due to the properties of the task structure where high quality methods tend to be uncertain and high certainty methods tend to have low quality. In this third case, the highest ranked alternative would be the same as the highest ranked in the certainty only case because it has the highest certainty to quality ratio.

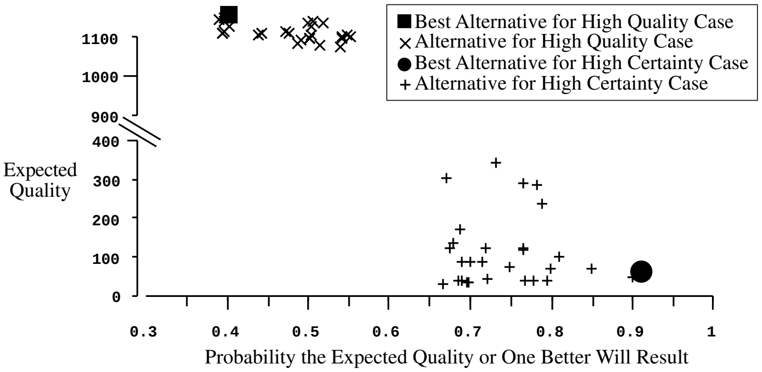


Figure 10: Alternatives Generated for Two Different Criteria Sets

As discussed earlier, in addition to the criteria driven role of uncertainty, the scheduler can also take a more active role in uncertainty reduction by generating alternatives that contain more than one way (other alternatives) to achieve various tasks. This redundancy flavored scheduling may serve to reduce uncertainty and it provides the scheduler with more options to consider. This is critical in some situations involving hard deadlines because in the event of a failure there is not always enough time left to try a different solution approach, i.e., once committed to a course of action, it is sometimes too late to reschedule and try again if a failure occurs. Consider a brief example. Figure 11 shows a task structure fragment, the relevant method attributes, and two schedules. The results generated by Task A are necessary for Task B and there is a hard deadline of 30 minutes. Schedule 1 contains no redundancy, having one method for achieving Task A and one for achieving Task B. Schedule 2 contains redundant methods for achieving Task B and uses a lower quality but more certain and faster method for achieving Task A. If Schedule 1 is executed and method A1 fails, 20 minutes are wasted and there is not time to reschedule and execute method A2 followed by either B1 or B2 prior to the deadline. Additionally, if method B1 fails there is also not time to reschedule and execute B2. However, if Schedule 2 is executed, we are as certain as possible that some results will be generated by the deadline because A2 is very certain and the less-certain-but-higher-quality B1 is followed by the more-certain-but-lower-quality B2. Considering uncertainty in conjunction with redundancies is clearly important in some situations. When the redundancy alternative generation feature is used, the alternatives that contain redundant activities are added

[12] Also applicable only in a multi-agent context.

to the alternative set and compared to the goal criteria in the same fashion as the non-redundant alternatives. Thus, the scheduler continues to focus processing on alternatives that best satisfice to meet the overall goal criteria – uncertainty does not dominate the evaluation mechanism unless so specified by the goal criteria.

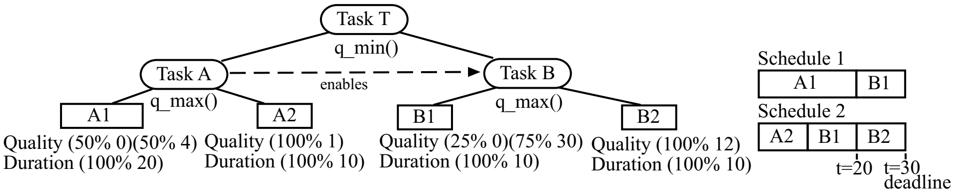


Figure 11: Redundancy Can Be Critical

It is important to note that the existence of a redundant method in a schedule does not mean that the redundant method will be executed every time. The execution of said method is dependent on the rescheduling triggers or envelopes associated with the schedule. The existence of the redundant method in the schedule does imply that the schedule can be executed from end-to-end without rescheduling to recover from particular errors. However, one of the main benefits of including redundancy in the schedule is analytical – it enables the scheduler to evaluate the performance characteristics of a problem solving episode that includes method failure and recovery instead of simply assuming no failure. When viewed in this light, redundancy is a very weak form of contingency planning and is related to the secondary contingency analysis algorithms presented in Section 5.

Modeling uncertainty improves and empowers other aspects of the scheduling process as well. In environments where rescheduling is undesirable the scheduler can use the probability distributions to design more fault tolerant schedules. For instance, if fault tolerance with respect to duration is desired, the scheduler can build schedules by estimating method execution times using the 95th percentile duration value rather than the expected value. In this situation, uncertainty about finish times still gets propagated throughout the schedule, but timing assumptions are based on a higher value that is by definition very certain.

The uncertainty representation can also improve the probability that little work is wasted in the event of a mid-schedule failure. Because of task interactions it is possible that a method failure anywhere in the schedule can void all the work done up to that point. Modeling uncertainty makes it possible for the scheduler to move the highly uncertain activities toward the front of the schedule, thus reducing the likelihood of doing work that is voided later in the schedule. This can be achieved through a new method rating heuristic that gives preference to methods that have some probability of failure and interact with other methods – or methods that have a probability of failure and are particularly important to the schedule. We will forgo further exploration of this idea in the context of the main scheduling process as these concepts have contributed to a secondary contingency analysis phase discussed in detail in Section 5.

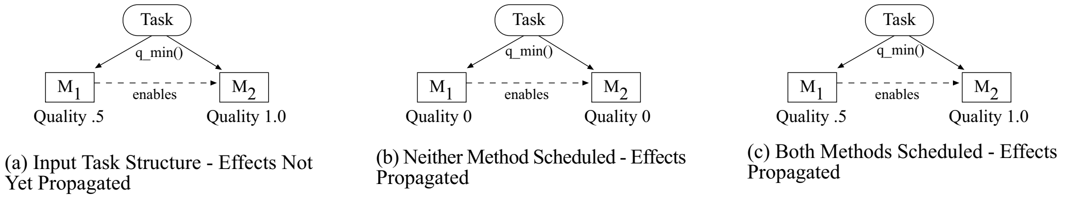
4.3 Impact of Uncertainty to the Computations and Schedule Models

The implications of the addition of uncertainty to the TÆMS modeling framework are not all positive – at least not from a computational expense standpoint. Maintaining and performing calculations with distributions is inherently more expensive than working with single expected values. Additionally, distribution sizes generally grow as computations progress. For example, combining two discrete probability distributions, where the distributions have n and m $(probability,\ value)$ pairs respectively, results in a distribution having $(n \times m)$ $(probability,\ value)$ pairs (though like values may be combined). While this does not change the combinatorics of the scheduling process, it adds significantly to the constant terms involved, even when the distributions are size-limited and compacted [13] periodically.

Another downside to the addition of uncertainty to TÆMS models, and its incorporation into the scheduling process itself, is that it invalidates a particular independence assumption that enables local evaluation of primitive actions. Said independence assumption simplifies calculations and saves considerably on the computational expense of reasoning about task interactions. The assumption is simply that the effects of any active nles can be accurately

[13] Compaction can lead to a loss of information and the introduction of estimation error into the computation. However, the estimation error is generally small and does not adversely affect the decision processes used in the scheduler.

reflected in the distributions of the node that is on the receiving end of the nles. Implementationally, this means that whenever the context changes, and nles be come active, or switch to an inactive state, the distributions on the recipient node are updated to reflect this state. With the addition of uncertainty to the task models, this assumption no longer holds.



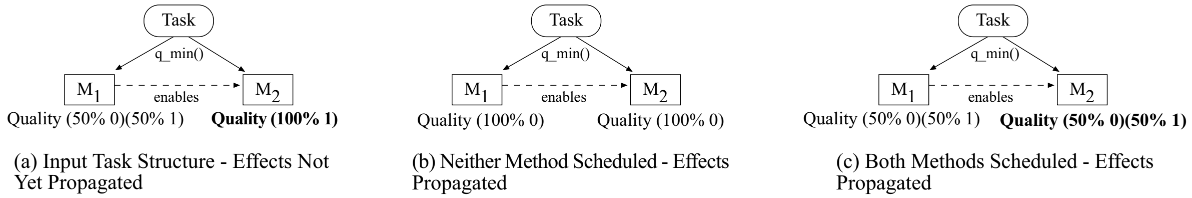
(a) Input Task Structure - Effects Not Yet Propagated

(b) Neither Method Scheduled - Effects Propagated

(c) Both Methods Scheduled - Effects Propagated

Figure 12: Independence Assumption Valid with Expected Values

Figure 12 illustrates the assumption under the expected value case. Figure 12(a) shows the input TÆMS task structure; the effects of interactions are not yet propagated to effected nodes. In the structure, method M_1 enables M_2 and the two methods are joined under the *q_min()* qaf; thus the quality of $Task$ is the minimum of the qualities of M_1 and M_2. Prior to scheduling either method, Figure 12(b), the expected quality of M_1 is zero, the expected quality of M_2 is zero, thus $Task$ also has an expected quality of zero. Once M_1 is scheduled, M_1's expected quality becomes .5. At this point, the enables nle between M_1 and M_2 becomes active and M_2's potential quality, that which can result if it is scheduled, becomes 1. Since there is no probability that M_1 may fail, M_2 is either enabled or it is not. When M_2 is scheduled, Figure 12(c), its quality reflects the assumption that the required input will be available and that M_2 will produce the expected result. In this case, $Task$'s quality is $min(.5, 1) = .5$ and is correct. This is the independence assumption at work; the same property holds for soft interactions like facilitation or hindering. Additionally, the property holds for chains of such relationships. Implementationally, this means that each time a method is scheduled, the effects of the outgoing nles can be reflected and propagated throughout the task structure and then the nle may be ignored. [14]



(a) Input Task Structure - Effects Not Yet Propagated

(b) Neither Method Scheduled - Effects Propagated

(c) Both Methods Scheduled - Effects Propagated

Figure 13: Independence Assumption Invalid with Uncertain Models

However, with the addition of uncertainty to the model nles are no longer binary, i.e., they are not simply active or not. Instead, there is some probability that they will be active and some probability that they will be inactive. Figure 13(a) shows the same task structure enhanced with the discrete probability distribution representation. Prior to M_1 being scheduled, there is no probability that M_2 is enabled and thus both M_1 and M_2 have zero expected quality as does $Task$, Figure 13(b). However, once M_1 is scheduled it may produce quality 50% of the time and fail 50% of the time. We reflect this possibility in the potential quality distribution of method M_2, i.e., if M_2 is scheduled, Figure 13(c), 50% of the time it will not have the required input and 50% of the time M_1 will succeed and produce the required input. The propagation of the probability of not having the required input is valid, but, the independence assumption no longer holds. Consider the quality of $Task$ if both methods are scheduled. M_1 and M_2 each fail 50% of the time, thus $Task$'s quality distribution is: $Task_{quality} = ((.25, min(0, 0)),\ (.25, min(0, 1)),\ (.25, min(1, 0)),\ (.25, min(1, 1)))$. After combining like values (zeros), the distribution becomes: $Task_{quality} = ((.75, 0),\ (.25, 1))$ and its expected value is .25. This is inaccurate because M_2 fails *iff* M_1 fails to produce the required result and M_1 fails to produce said result

[14] This is not quite accurate. During scheduling, many different contexts are explored and the computations are repeated many times. However, when constructing a given schedule, once a method is scheduled, the computations do not need to be repeated under the independence assumption.

50% of the time. Thus, $Task$ should only fail to obtain quality 50% of the time and the remainder of the time it should obtain $min(1,1) = 1$, resulting in an expected value of .5. With the addition of uncertainty, and the representation of some probability of failure, the independence assumption no longer holds but instead leads to over-emphasis on failure effects throughout the task structure (M_1 and M_2 may be widely distributed in the structure).

The nle-effect-reflection type of calculation is performed an enormous number of times during scheduling. For a moderately sized task structure, it is not uncommon to perform hundreds of thousands of distribution combination operations in a single scheduling episode. To maintain efficiency, the independence assumption is left in place during estimation, approximation, and method sequencing. However, once the set of candidate schedules is produced, each schedule is re-evaluated using a tree-based in-context analysis approach that corrects the estimation errors in the computation, Figure 14. The complexity of the tree-based analysis is driven by the frequency of method failure within a given schedule and thus is occasionally too expensive even when used in this limited context.

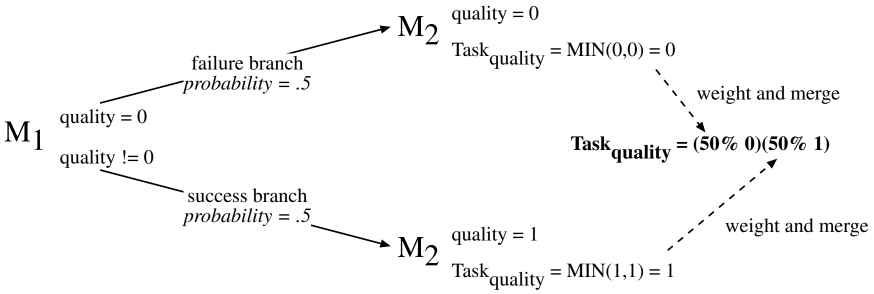


Figure 14: Accurate, Contextual, Execution Tree Computation is Expensive

The addition of uncertainty also affects schedule construction and reasoning about start times, finish times, cost limits, and deadlines. Since methods may have a range of possible durations, as schedules are constructed, the uncertainty associated with the durations must be propagated – methods no longer have single finish times but instead have distributions of possible finish times. Additionally, since methods are serialized when scheduled, the uncertainty of the methods scheduled before a given method affect its start time (a distribution) and consequently also its finish time distribution. This complicates matters when determining whether or not a particular method will complete before a deadline, or whether or not a result will be available to satisfy a commitment made to another agent by the desired time. We leverage the improved models in these situations to reason about the probability of violating or satisfying a particular constraint.

Consider the deadline case; if a method produces a result after a hard deadline, the result is considered valueless and thus the method's quality result is zero. When reasoning about deadlines from an uncertain perspective, we reflect the possibility that a given method will exceed its deadline by modeling the effects of this violation in its quality distribution. For example, Figure 15, if M_x has a 10% chance of exceeding its deadline, the densities of all the members of its quality distribution are multiplied by 90% (thus re-weighting the entire distribution) and a new density / value pair is added to the distribution to reflect the 10% chance of returning a result after the deadline. The leftmost histogram describes M_x's expected finish time, the middle histogram describes M_x's unmodified quality distribution, and the rightmost figure shows the modified quality distribution after re-weighting and merging with the new $(10\%,\ 0\ quality)$ pair. This quality-based reflection is important because it improves the scheduler's ability to reason about hard deadlines.

4.4 Scheduling to Reduce Uncertainty within Design-to-Criteria

To illustrate the benefit of modeling and using uncertainty in the main Design-to-Criteria process, let us consider the problem of custom building schedules for two different clients from a moderately complex task structure. The task structure has methods that fall into three general categories. 1) Methods that have high expected quality values also tend to take longer and are highly uncertain in both the quality and duration dimensions. 2) Methods that have low expected quality also tend to take less time to execute and are more certain in both the quality and duration dimensions.

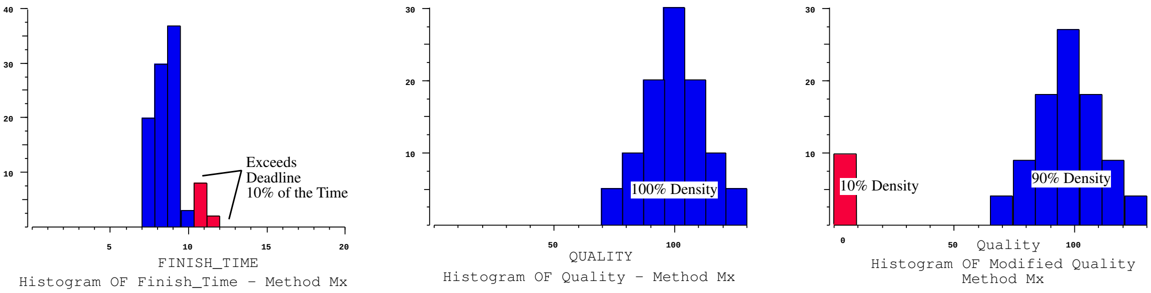


Figure 15: Reflecting Probability of Missing Deadline in Method Quality

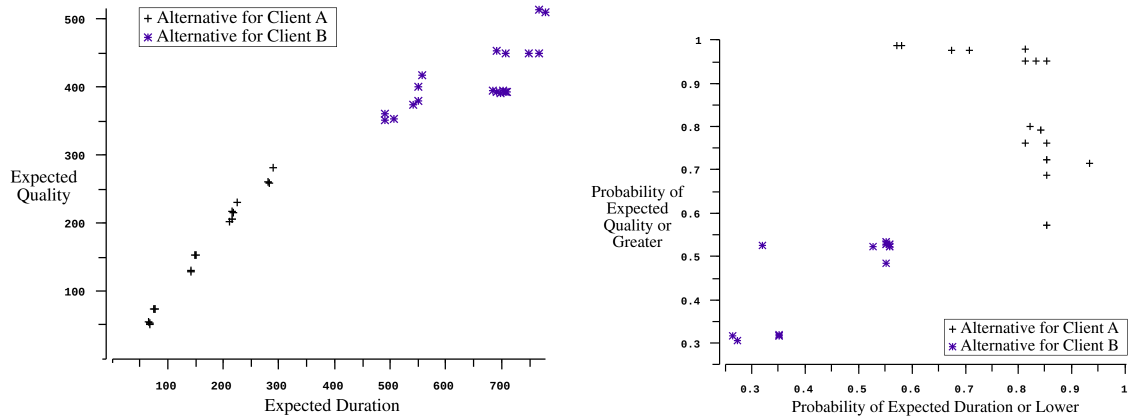
3) Methods that have medium expected quality also take a moderate time to execute and are moderately certain.

The high-quality-but-uncertain methods model information gathering tasks that are risky but also have a probability of a large information pay-off. For example, methods of this type may find information about a software product by submitting multiple queries to Infoseek and Altavista, going to the URLs, retrieving multiple documents from each site, and processing them. As the information located can range from useful new information with wide-scale ramifications to utterly useless information that is not relevant, there is the probability of big pay-offs and also the probability of zero or poor results. Since methods of this type use a large amount of active web search on sites that are unknown *a priori*, predicted duration is also long and uncertain. The low-quality-but-more-certain methods model information gathering tasks where information is retrieved from individual sites that are known and modeled. Since the information is predicted to be fairly narrow in scope, these methods lack the potential for big pay-offs, however, since the methods search only one site and the site in question is modeled, durations are short and fairly certain. The middle-quality-middle-certainty methods employ combinations of these behaviors.

Since the first client, Client A, is planning other activities based on the predicted outcome of schedule execution, this client is interested in both schedule raw-goodness and schedule certainty. In the raw-goodness slider bank the quality slider is set to 75% and the duration slider set to 25%, i.e., overall quality is 3 times more important than overall duration. In the uncertainty bank the quality and duration sliders are each set to 50%, meaning that certainty about the estimated quality and certainty about the estimated duration are equally important. The meta slider for raw-goodness is set to 40% and the meta slider for uncertainty is set to 60%, denoting that uncertainty reduction is 1.5 times more important than raw schedule goodness. Unlike Client A, Client B has much simpler needs and is only interested in raw-goodness. As with Client A, the raw-goodness quality slider for this client is set to 75% and the raw goodness duration slider is set to 25%. The meta-slider for raw goodness is set to 100% denoting that raw goodness is the only issue of importance to this client.

Figure 16(a) shows the expected quality and expected duration of the top-level alternatives generated for Clients A and B; intermediate alternative sets were pruned according to the client's goal criteria as discussed previously. Despite both clients setting the raw quality and duration sliders to the same values, Client B's alternatives always have higher expected quality and higher expected duration than Client A's. Since neither client is using hard deadlines, this is attributable to Client A's emphasis on certainty about quality and certainty about duration. Figure 16(b) tells the rest of the story. As Client A put 60% of the overall weight on certainty in the quality and duration dimensions, the alternatives generated for Client A trade-off between raw quality, raw duration, quality certainty, and duration certainty, rather than just trading-off quality and duration. Figure 16(b) also shows the price of B's high expected quality – the expected values are also predicted to be much more uncertain than those of Client A.

The quality and duration attributes of the schedules produced from a subset of these alternatives are similar to the attributes of the alternatives. In this case, the estimates contained in the alternatives are fairly good indicators of the schedules produced from the alternatives. This indicates that subtask interactions in the alternatives generated and targeted for scheduling were fairly simple and generally involved hard-precedence constraints. In keeping with intuitions, the highest rated schedule for Client B is that which has the highest expected quality with respect to duration. However, Client A's "best schedule" has a reasonably good quality for its expected duration and a high degree of certainty about its expected quality and duration values.



(a) Alternatives for A and B

(b) Probability of Expected Values of Alternatives

The quality and duration results of executing the best schedules for each client thirty times are shown in Figure 16. Whereas Client A's executions produced a tightly spaced set of quality and duration values, Client B's highly uncertain schedule produced a wide range of results. Of the thirty runs, Client A's results meet or beat expectations in the quality dimension 90% of the time, in the duration dimension 50% of the time, and in both the quality and duration dimensions 50% of the time. In contrast, Client B's results only meet or beat quality expectations 63% of the time, duration expectations 16% of the time, and both dimensions combined 13% of the time. Additionally, the uncertainty in B's quality dimension incurred more rescheduling because of methods failing to return any results (problematic because of task interactions). On average, B's schedule required rescheduling 2.1 times per each execution, with a variance of .71, whereas A's only required 1.2 reschedulings on average with a variance of .21. The 25% trimmed mean brings out the contrast even more – B's rescheduling average remains 2.1 but A's 25% trimmed mean drops to 1.0, denoting no rescheduling during execution.

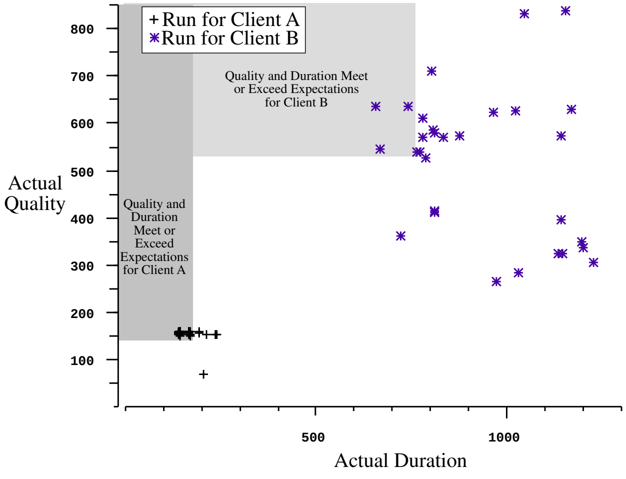


Figure 16: Execution Results for A and B

5 Uncertainty-based Contingency Analysis

In the previous sections we explored uncertainty as it is integrated into the standard Design-to-Criteria scheduling methodology. However, in situations where hard deadlines exist, a mid-schedule failure may preclude recovery via rescheduling because sufficient time does not remain to explore a different solution path. In these situations, a stronger analysis that considers the existence of possible recovery options may lead to a better choice of schedules. To address such situations, we have developed a contingency analysis methodology that functions as an optional back-end on the Design-to-Criteria scheduler.

The contingency analysis algorithms operate by examining the highly-rated candidate schedules produced by the scheduler and exploring failure / recovery scenarios for each schedule in the set. These secondary analysis tools also perform more detailed reasoning about the placement of methods within a schedule in light of the existence of recovery options. For example, recovery for a given schedule may be possible *iff* some critical method $m_{critical}$ is performed first rather than second. The standard scheduler is weakly biased toward moving uncertain methods earlier in the schedule, but the determination is *local,* based only on the attributes of the method in question, whereas the method movement explored in the contingency analysis also takes into account the benefits of method movement from a recovery perspective.

This underscores the primary difference between the use of uncertainty in the main Design-to-Criteria scheduling process and its use in the secondary contingency analysis algorithms. To address resource limitations and to produce schedules in interactive time, Design-to-Criteria builds and evaluates schedules in an independent fashion – the possibility of recovery from a particular failure is not considered by the main scheduling process. This is because determining the existence of a recovery option requires more than simply finding an action to replace the failure; indeed because of task interactions and the combinatorics of TÆMS models, the process of evaluating recovery options fully may require significant computational expense, e.g., trying all possible *alternative* ways in which a task might be achieved ($O(2^n)$).

This is true in general *contingency planning* as well [5]. In contrast, in the secondary analysis algorithms, we perform more detailed, contextual, schedule analysis based on the availability of recovery options and the possibility of failure at key points. This analysis is more expensive, but, in some situations, the added expense is warranted. For example, the process of determining a schedule for a world-class telescope does not have to be particularly timely, as the instrument is unused during day light hours, but evening observation time is too valuable to waste. In this situation, a detailed analysis that considers recovery options is worthwhile.

The MDP-based optimal meta-controller considers every possible outcome of the executable action and the implications on achieving the high level goal within the given criteria. Hence, while constructing a policy, the optimal controller evaluates every potential critical (failure-prone) region and prescribes the appropriate (contingent) action whe necessary. It does a thorough analysis of all possible critical situations and hence carries the overhead that goes with such an analysis. Our heuristic-based contingency analysis is an approximation of the optimal controller since it takes into account only *high impact* CTER's and has a relatively local view of implications of method outcomes compared to that of the optimal controller. In Section 6, we compare the performance of the contingency enhanced DTC scheduler to that of the optimal policy.

In this section we discuss contingency scheduling issues and formalize five different measures of schedule *robustness*, where robustness describes the quantity of recovery options available for a given schedule. In Section 6 we then present experiments comparing the use of the contingency algorithms to the standard Design-to-Criteria scheduling approach.

This work in contingency analysis of schedules is closely related to recent work in conditional planning. However, the planning-centric research focuses on solving problems which involve uncertainty by probabilistic reasoning about actions and information on the value of planning for alternative contingencies [13, 23] and using utility models [18]. Other approaches use partial Markov decision processes and decision theoretic planning approaches [4, 8] which prune the search space using domain-specific heuristic knowledge. [28] describes a partial-order planner called *Mahinur* that supports conditional planning with contingency selection. The authors concentrate on two aspects of the problem, namely, planning methods for an iterative conditional planner and a method for computing the negative impact of possible sources of failure. Our work addresses similar questions within the Design-to-Criteria application domain, namely:

1. *How can we effectively predict the performance of a schedule when there is uncertainty in the performance of methods in the schedule?*
2. *What are the different approximations to the execution-time performance measure and when is a specific approximation appropriate?*

[5] discusses an algorithm for a specific domain namely a real telescope scheduling problem where the stochastic actions are managed by a splitting technique. Here the Just-In-Case scheduler pro-actively manages duration uncertainty by using the contingent schedules constructed by analyzing the problem using off-line computations. Our contingency scheduling research differs from previous work in the following ways:

1. The contingency analysis algorithms use the Design-to-Criteria scheduler to explore mainly the "good" portions of the schedule solution space – that is those schedules that best address the client's design criteria. This serves to constrain the computation and reduces the combinatorics from their general upper bounds. More importantly, the algorithm presented here is amenable to future research in bounding the algorithm directly, which would enable the contingency analysis approach to operate in interactive time, as does the underlying Design-to-Criteria scheduler.
2. Contingency analysis takes place in the context of the multi-dimensional goal criteria mechanism used in Design-to-Criteria. Thus the analysis approach is fully targetable to different applications, e.g., situations where quality should be traded-off to obtain lower cost accompanied by a hard deadline, or situations in which quality should be maximized within a hard deadline.
3. Our algorithm takes advantage of the structural properties of the input problem. Namely the TÆMS task structure representation is used to constrain the analysis process and to help limit the exploration of the search used to locate recovery options. This is in contrast to a simple exploration of all primitive actions without regards for interactions or for how the actions relate to achieving the overall goal.

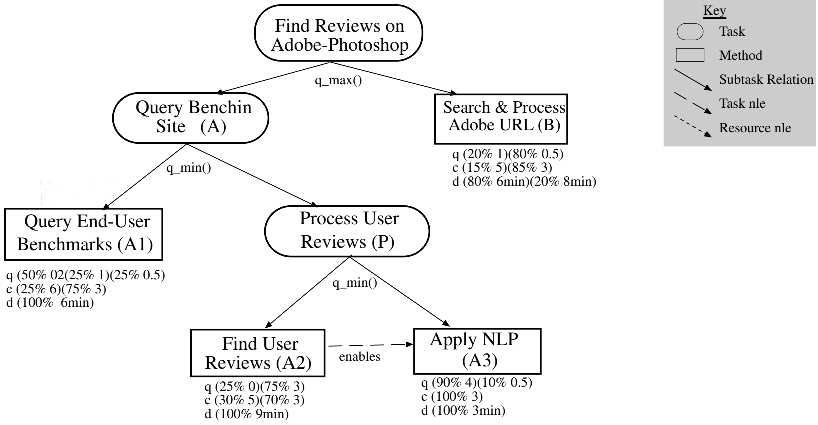


Figure 17: Gather review information on Adobe Photoshop.

In the introduction, we used a simple example described in Figure 3 to motivate the importance of uncertainty analysis. However, the characteristics of the example and its criteria were such that there were no recovery options possible in the event of failure of a method. We now consider another example described in Figure 17 which has the characteristics required to clearly illustrate the power of contingency analysis. In the discussion that follows, we use the latter example to exhibit the power of our heuristic-based contingency analysis. We show that the best schedule selected by the DTC scheduler has no recovery options in the event of failure while the MDP-based optimal policy as well as the best schedule selected by the contingency-enhanced DTC scheduler has built-in contingencies for recovery from failure.

The top-level task in Figure 17 can be achieved by either completing task *Query-Benchin-Site (A)* successfully or executing the method *Search-Adobe-URL (B)*, or both. If both A and B are executed the maximum quality of these two is the quality propagated to the parent node (per the $q_max()$ qaf). The quality, cost and duration distributions for the executable methods denote expectations about method performance. For instance, the quality distribution of method *End-User-Benchmarks* indicates that it achieves quality value of 2 with probability 0.5, quality of 1 with probability 0.25 and 0.5 with probability of 0.25. Lets assume The client design criteria is to maximize quality within a hard deadline of 18 minutes.

The MDP-based optimal policy for the above problem is shown in Figure 18. The policy suggests the method sequence $\{$*FindUserReviews, UserBenchMarks, ApplyNLP*$\}$ (A2,A1,A3) as the best schedule when method *FindUserReviews* achieves non-zero quality and $\{$*FindUserReviews*, *SearchAdobeURL*$\}$ (A2,B) would be an alternate schedule in the event of *FindUserReviews*'s failure to achieve quality.

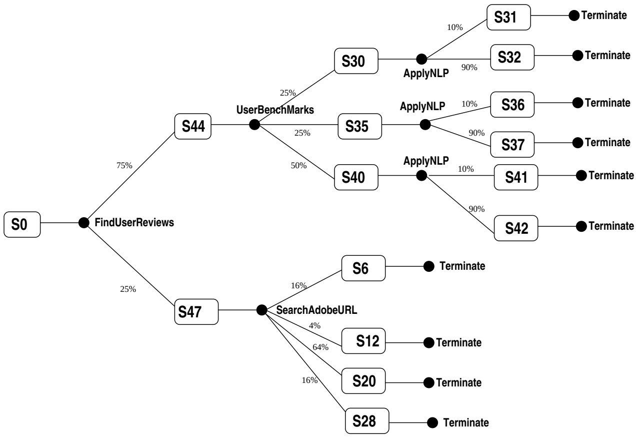


Figure 18: Optimal policy for Gather-Review-Information-on-AdobePhotoshop

The Design-to-Criteria scheduler first enumerates a subset of the alternatives that could achieve the high level task. A subset of these alternatives are selected and schedules are created using the one-pass method-ordering techniques identified in Section 4. The set of candidate schedules are then ranked using the multi-dimensional evaluation mechanism [36] which compares the schedules' statistical attributes to the client design criteria.

We will use the term *expected lower bound* (ELB) to denote a slightly modified schedule utility rating returned by the standard Design-to-Criteria scheduler. In the ELB computation, the standard utility value associated with the schedule is computed without the relative scaling components discussed in Section 4.1; this enables comparison between the ELB for a schedule belonging to one set, e.g., S_1, and a schedule belonging to a different set, S_2. For the purposes of illustration simplicity, we will discuss the ELB in this document as being directly related to the expected quality of a given schedule, i.e., in this document, the ELB *is* the expected quality of a given schedule assuming no rescheduling. In terms of the design criteria described in Section 4.1, this is equivalent to a client specifying a preference for maximizing quality within a given deadline – no weight or value are given to any of the other criteria dimensions. The algorithms presented in the following sections operate on more interesting criteria settings, but, the analysis is more easily understood if the metrics are cast in terms of expected qualities rather than a multi-dimensional objective / utility function.

For the example in Figure 17, the two possible schedules are $\{$A1,A2,A3$\}$ and $\{$B$\}$. Figure 19 describes the computation of the ELB for the schedule $\{$A1,A2,A3$\}$. Consider the first entry in the table. It describes the case when method A1 achieves a quality of 2, which occurs with a probability of 0.5 as described in the TÆMS task structure. Method A2 achieves a quality of 0 with probability 0.25. [15] The probability of the methods achieving these qualities in a single execution is 0.125, given in column 4. The expected quality of the schedule $\{$A1,A2,A3$\}$ is 0 in this case, described in column 5. The duration and cost distributions and their expected values are computed in a similar fashion.

[15] Failure of A2 (where quality= 0) results in zero quality for the schedule due to the way in which the task structure is defined, i.e., under $q_min()$ qafs, failure results in zero quality for the parent task as well. Hence the quality of A3 is a not a determining factor and is represented by nil.

A1	A2	A3	Frequency	Quality
50% 2	25% 0	nil	5%*25%=12.5%	0.0
50% 2	75% 3	90% 4	33.75%	2.0
50% 2	75% 3	10% 0.5	3.75%	0.5
25% 1	25% 0	nil	6.25%	0.0
25% 1	75% 3	90% 4	16.875%	1.0
25% 1	75% 3	10% 0.5	1.875%	0.5
25% 0.5	25% 0	nil	6.25%	0.0
25% 0.5	75% 3	90% 4	16.875%	0.5
25% 0.5	75% 3	10% 0.5	1.875%	0.5

Figure 19: Each row represents a possible permutation of the quality distributions of methods A1, A2, A3 in schedule {A1,A2,A3}. The first three columns represent the possible expected quality values achieved by each of the methods A1, A2, A3. The fourth column shows the probability of the particular quality distribution combination occurring and the last column shows the final expected quality of the schedule.

The ELBs for schedules {A1,A2,A3} and {B} are as follows:

1. {A1,A2,A3}: ELB: 0.97 (Expected Quality)
   Quality : (25% 0.0) (24% 0.5) (17% 1.0) (34% 2.0)
   Duration : (100% 18)
2. {B}: ELB: 0.6
   Quality : (20% 1) (80% 0.5)
   Duration: (80% 6) (20% 8)

Since {A1,A2,A3} has the highest ELB (indeed, the highest rating using the standard normalized utility functions), it is chosen and executed. Suppose A1 executes successfully, but A2 fails (i.e. it results in 0 quality), which it does 25% of the time. Then A3 cannot be executed because it is not enabled (A2 failed) but there is no time left to reschedule and attempt method {B} because there is not sufficient time to execute method B before the deadline.

Because of the one-pass low-order polynomial method sequencing approach used by the scheduler to control scheduling combinatorics, the standard Design-to-Criteria scheduler will only produce one permutation of the methods A1, A2, and A3. However, if the scheduler did produce multiple permutations, the schedules {A1,A2,A3} and {A2,A1,A3} would receive the same expected lower bound value. Hence the contention is that there is no difference in performance between the two. However with more detailed evaluation of the schedules, it is clear that {A2,A1,A3} allows for recovery and contingency scheduling which schedule {A1,A2,A3} does not permit for the given deadline. If {A2,A1,A3} is the schedule being executed and A2 fails, there is time to schedule method {B} and complete task TG1. This clearly implies that schedule {A2,A1,A3} should have a better expected performance rating than {A1,A2,A3} as the schedule {A2,A1,A3} includes the recovery option from failure in its structure.

5.1 Critical Task Execution Regions and the Approximate Expected Upper Bound

In our example, task A2 has an enables non-local effect as well as a 25% chance of failure within its distribution. We hence predict that task A2 could potentially be a *critical task execution region* (CTER). A *CTER* is a method that has the potential to seriously degrade the performance characteristics [16] of the overall schedule if it should fail. We will use the term *approximate expected upper bound* (AEUB) to denote the expected quality of schedules that are computed with the *CTER*'s criticality removed. The AEUB is defined formally in the next section – the discussion here is intuitive. Removing the possibility of failure in the AEUB enables us to better understand the implications of the potential *CTER* on the rest of the schedule. For this example, let us remove the failure possibility from the performance characterization of A2 and replace method A2's 25% chance of quality 0 with the expected value of the

[16] A method could have uncertainty in its performance characteristics but this uncertainty might not affect the method's outcome significantly. We restrict our classification of methods as CTER's to those which have the most impact to the schedule performance under limited duration constraints.

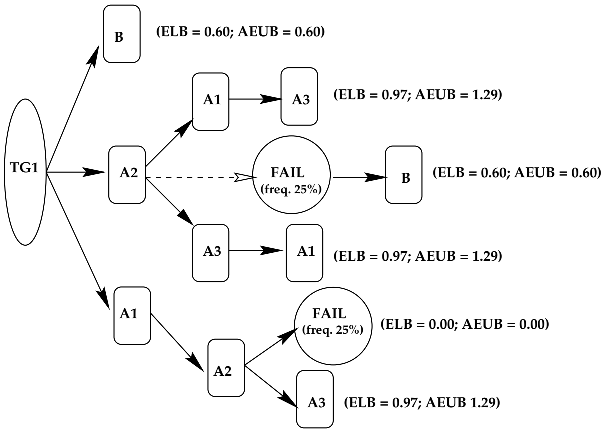

B
(ELB = 0.60; AEUB = 0.60)
A1
A3
(ELB = 0.97; AEUB = 1.29)
TG1
A2
FAIL
(freq. 25%)
B
(ELB = 0.60; AEUB = 0.60)
A3
A1
(ELB = 0.97; AEUB = 1.29)
A1
FAIL
(freq. 25%)
(ELB = 0.00; AEUB = 0.00)
A2
A3
(ELB = 0.97; AEUB 1.29)

Figure 20: Schedule Options for IG Task (Figure 17) where Ratings are Expected Qualities

distribution. Method A2 hence is assigned a quality of 3, with a probability of 1, i.e. for method A2, Q (100% 3). The Design-to-Criteria scheduler is reinvoked with the modified task structure and rescheduled. The following are the AEUBs (expected qualities that result with the possibility of failure removed) returned by the scheduler.

1. $\{A1, A2^{success}, A3\}$: AEUB 1.29
   Quality : (32% 0.5)(23% 1.0)(45% 2.0)
   Duration: (100% 18)
2. {B} : AEUB 0.6
   Quality: (20% 1) (80% 0.5)
   Duration: (80% 6) (20% 8)

The new AEUB statistic describes performance expectations if failure is not possible. The relationship between the AEUB and the ELB is a clue to the importance of the potential *CTER* to the overall schedule. In this case, the schedule {A1,A2,A3} now has an expected quality value of 1.29. The $\frac{1.29-0.97}{0.97} * 100 = 33$ % improvement in quality with respect to the ELB is significant. This 33% improvement in quality confirms that the possibility of failure in method A2 significantly decreases the rating of schedule {A1,A2,A3}. The next step is to consider the optional schedules for the original task structure to neutralize the effect of this *CTER*.

The tree structure in Figure 20 presents all possible scheduling options, including recovery scenarios, that meet the hard deadline of 18 minutes. From this diagram, we see that schedule $\{A1, A2, A3\}$ does not have an option to reschedule and still meet the deadline if method A2 fails. Thus we consider a simple reordering of schedule $\{A1, A2, A3\}$ which is $\{A2, A1, A3\}$. To assess the effects of rescheduling when A2 fails on this schedule {A2,A1,A3}, we combine the ratings for schedules $\{A2^{success}, A1, A3\}$ and $\{A2^{failure}, B\}$ based on their likelihoods of occurrence. So a schedule starting with A2 gets a rating of $\frac{75}{100} * 1.29 + \frac{25}{100} * 0.60 = 1.1175$. We use a similar analysis to get the values of schedules starting with A1 $= \frac{75}{100} * 1.29 + \frac{25}{100} * 0 = 0.9675$ and B $= 1 * 0.60 = 0.60$ This type of schedule evaluation is what we call the *approximate expected bound* (AEB), which is formally defined in the next section. Note that with this detailed analysis it is clear that schedule $\{A2, A1, A3\}$ has better expected performance than $\{A1, A2, A3\}$. However, the ELB computation of the Design-to-Criteria scheduler returns an identical ELB for both $\{A1, A2, A3\}$ and $\{A2, A1, A3\}$ as it does not take into account the recovery options present within $\{A2, A1, A3\}$. This leads us to believe that the ELB perhaps is not the most appropriate performance measure for all task structures, particularly where hard deadlines or cost limits (in contrast to soft preferences) are important and failure is possible.

Figure 21 illustrates this concept. The figure contains two histograms, one displaying the quality that results from executing the highest rated schedule produced by the standard scheduler, namely $\{A1, A2, A3\}$, and one displaying the quality that results from executing the modified schedule $\{A2, A1, A3\}$. The results are presented in a left to right fashion. In each case the chosen schedule was executed 100 times in an unbiased simulation environment in which

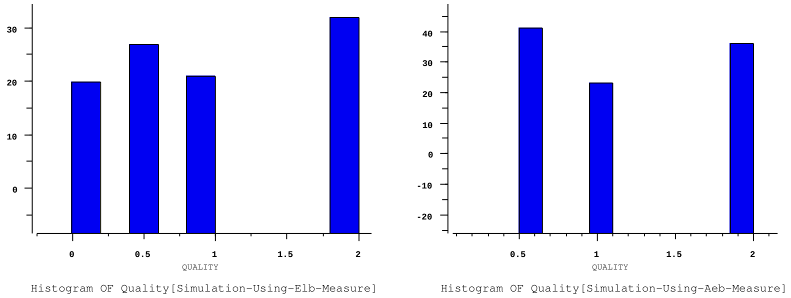


Figure 21: Performance of Expected Lower Bound and Approximate Expected Bound Selected Schedules

the execution results are determined by sampling from the distributions associated with the given methods.[17] Recall that the standard scheduler will give these schedules identical ratings as it does not consider recovery options. The execution results are consistent with the claim that the schedules are not actually equivalent. The schedule produced by the standard scheduler fails to generate quality about 20% of the time and the mean resultant quality is 0.98. In comparison, the reordered schedule never produces a zero quality result, as it leaves time for recovery, and its mean resultant quality is thus significantly higher, namely 1.96.

5.2 Performance Measures

In this section we formalize a general theory relating to the contingency planning concepts discussed in the previous section. The question we strive to answer formally here is the following: *What performance measure is the most appropriate estimator of the actual execution behavior of a schedule given the possibility of failure?* Our basic approach is to analyze the uncertainty in the set of candidate schedules to understand whether a better schedule can be selected or an existing schedule can be slightly modified such that its statistical performance profile would be better than that normally chosen by the Design-to-Criteria scheduler. We accomplish this analysis through the use of several performance measures. As mentioned earlier, contingency planning is a heuristic approach. Thus these performance measures are approximations of an optimal policy computation. Prior to presenting the measures, a few basic definitions are needed:

1. A schedule s is defined as a sequence of methods $(m_1, m_{2,}..m_{n-1}, m_n)$.
2. Each method has multiple possible outcomes, denoted m_{ij}, where j denotes the j'th outcome of method m_i. This is part of the TÆMS definition of methods or primitive actions. Though the examples generally present methods as having quality, cost, and duration distributions, methods actually may have sets of these distributions where each set is one possible outcome. For example, if method m may produce two classes of results, one class that is useful by method m_1, and one class that is useful by method m_2, method m will have two different possible outcomes, each of which is modeled via its own quality, cost, and duration distributions. Additionally, these different outcomes will have different non-local effects leading from them to the client methods, m_1 and m_2 respectively.
3. Each outcome is characterized in terms of quality, cost, and duration, via a discrete probability distribution for each of these dimensions and each outcome has some probability of occurrence.
4. m_{ij}^{cr} is a *CTER* when the execution of m_i results in outcome j which has a value or set of values characterized by a high likelihood that the schedule as a whole will not meet its performance objectives. For instance, m_{ij} is a CTER if the probability of the quality of m_{ij} being zero is non-zero.
5. A schedule s could have zero, one or more CTER's in it. A general representation of such schedule with at least one CTER would be $s^{cr} = (m_1, m_{2,}..m_{ij}^{cr}..m_{kl}^{cr}...m_{no}^{cr}..m_{n-1}, m_n)$.
6. f_{ij}^{cr} is the frequency of occurrence of m_i's , j'th outcome where m_{ij} is a CTER.
7. $\overline{m_i^{cr}}$ is m_{ij}^{cr} with its current distribution being redistributed and normalized after the removal of its critical outcome. In other words, the criticality of m_{ij}^{cr} is removed and the new distribution is called $\overline{m_i^{cr}}$.

[17] This is in contrast to other experiments done with the scheduler, not included in this work, in which the environment is biased in some way or in which the agent sees an imperfect or *subjective* view of some *objective* task structure.

8. If s^{cr} = $(m_1.., m_{i-1}, m_{ij}^{cr}, m_{i+1}, .m_{kl}^{cr}..m_{no}^{cr}..m_{n-1}, m_n.)$, then
$\overline{s_i^{cr}}$ = $(m_1.., m_{i-1}, \overline{m_i^{cr}}, m_{i+1}, ..m_{kl}^{cr}..m_{no}^{cr}..m_{n-1}, m_n.)$
$\overline{s_k^{cr}}$ = $(m_1.., m_{i-1}, \overline{m_i^{cr}}, m_{i+1}, .\overline{m_k^{cr}}..m_{no}^{cr}..m_{n-1}, m_n.)$ and
$\overline{s^{cr}}$ = $(m_1.., m_{i-1}, \overline{m_i^{cr}}, m_{i+1}, .\overline{m_k^{cr}}..\overline{m_n^{cr}}..m_{n-1}, m_n.)$

The five statistical measures that aide in detailed schedule evaluation are:

Expected Lower Bound (ELB) The expected lower bound rating, of a schedule s, is the performance measure of a schedule execution without taking rescheduling into consideration [37]. It is a expected rating because it is computed on a statistical basis taking quality, cost and duration distributions into account, but ignoring the possibility of rescheduling. As mentioned previously, in this paper, to simplify presentation of the algorithms we will concentrate on the case in which the ELB is only the expected quality of a given schedule. In the general case, the ELB is the utility value generated by the computations presented in Section 4.1 with the relative scaling aspect of the computation removed.

Approximate Expected Upper Bound (AEUB) The AEUB is the statistical schedule rating after eliminating all regions where rescheduling could occur. The assumption is that there are no failure regions and hence the schedule will proceed without any failures and hence no rescheduling will be necessary. The following is a formal definition of AEUB:

Suppose m_{ij}^{cr} is a CTER in the schedule $s = (m_1..m_n)$ and it occurs with frequency f_{ij}^{cr}. Let $\overline{s_i^{cr}} = (m_1, m_2..\overline{m_i^{cr}}..m_n)$.

If $\frac{ELB(\overline{s_i^{cr}})-ELB(s)}{ELB(s)} \geq \alpha$, then m_{ij} is a *CTER*, where α is a percentage value that determines when a region should be classified a *CTER* and thus a candidate for more detailed analysis. The value of α is contextually dependent and should be specified by a scheduler client. For instance, if saving on computational expense is more important to the client than high certainty, α should be high, and thus the threshold for *CTER* classification is also high. However, if certainty is paramount, then α should be low, indicating that any significant change in the ELB should be explored.

For our information gathering example, we see that $\frac{ELB(\{\overline{A2},A1,A3\})-ELB(\{A1,A2,A3\})}{ELB(\{A2,A1,A3\})} \geq 0.3$. Hence there is at least an 30% increase in the schedule rating if the likelihood of failure of A2 is removed.

When this computation is done on an entire schedule for all of its *CTER's*, we call it the Approximate Expected Upper Bound. Generalizing this formula for k *CTER's* $m_{i_1 j_1}...m_{i_k j_k}$,

$AEUB(s) = ELB(m_1...m_{i_1-1}, \overline{m_{i_1}^{cr}}..\overline{m_{i_2}^{cr}}......\overline{m_{i_k}^{cr}}...m_n)$.

The AEUB is thus the best rating of a schedule on an expected value basis without any rescheduling.

In contrast, the optimal policy describes the next best action based on the execution characteristics of the last action taken. Hence rescheduling is built within the policy and failure regions are are not ignored in the state expansion. Hence the performance characteristics of the optimal policy is more exact than the ELB and AEUB.

Optimal Expected Bound (OEB) The OEB is the schedule rating if rescheduling were to take place after each method execution. So the first method is executed, a new scheduling subproblem which includes the effects of the method completion is constructed and the scheduler is re-invoked. The first method in this new schedule is executed and the steps described above are repeated. Hence the optimal[18] schedule is chosen at each rescheduling region. For complex task structures, the calculation would require a tremendous amount of computational power and it is unrealistic to use it for measuring schedule performance in a real system.

In most situations, $ELB(s) \leq OEB(s) \leq AEUB(s)$, since the $OEB(s)$ is based on recovery from a failure while $AEUB(s)$ assumes no failure.

Since our MDP based optimal policy does not suffer from instantiation effect(possible negative influence by the choice of the initial method for execution), the performance characteristics of the optimal policy should be the same as that of the OEB.

[18] "Optimal" in this case is meant in a satisficing fashion. In the context of Design-to-Criteria, the "best" schedule for a given task structure is not guaranteed to be optimal as the combinatorics prevent an exhaustive search. As it is used here, optimal means the best possible schedule within the space searched by Design-to-Criteria.

Expected Bound (EB) Let m^e_{ij} be the set of actual quality, cost, duration values when method m_{ij} is executed. After each method execution the schedule is re-rated. If for some schedule s = $(m_1, m_2..m_i..m_n)$,and $ELB((m_1...m_n)) \gg ELB((m^e_{1j}, m^e_{2k}...m^e_{il}, m_{i+1}..m_n))$, i.e. the actual execution performance of a schedule is below expectation, then a new schedule is constructed based on the partially complete schedule $\{m^e_{1j}, m^e_{2k}, ...m^e_{il}\}$.

So the EB is the schedule rating when rescheduling occurs only when there is a possibility for the partial execution of the current schedule will fail to meet expected criteria as a result of the outcomes of methods already executed. This computation, like the OEB, will require extensive computational power. Again in most situations, $ELB(s) \leq EB(s) \leq OEB(s) \leq AEUB(s)$.

The optimal policy generated by the MDP based method inherently handles small effects(accumulation of below expectation performance) and instantiation effects. Hence the policy's performance measure will be as good as if not better than that of the EB.

We are currently working on an experimental evaluation of this performance measure which will help determine whether incremental effects of non-critical outcomes could lead to a critical state. These evaluations are too preliminary to be discussed in this paper.

Approximate Expected Bound (AEB) It is the schedule rating with rescheduling only at *CTER's* and using expected lower bound of the new stable schedule for methods following the *CTER*. This is limited contingency analysis at *CTER's*.

Consider a schedule s of n methods s= $(m_1, m_2..m_i..m_n)$. Now suppose m_{ij} is a *CTER* with a frequency of occurrence of f_{ij}. In order to compute the AEB of the schedule, we replace the portion of the schedule succeeding m^{cr}_{ij}, which is $m_{i+1}, m_{i+2}, ...m_n$ by $l_{i+1}, l_{i+2}......l_k$ if there exists a $l_{i+1}, l_{i+2}......l_k$ such that

$ELB(m_1...m^{cr}_{ij}, l_{i+1}...l_k) \geq ELB(m_1...\overline{m^{cr}_i}, m_{i+1}...m_n)$.

The Approximate Expected Bound for this instance is computed as follows:
$AEB_{ij}(m_1, ...m_n)$=$ELB(m_1...\overline{m^{cr}_i}, m_{i+1}..m_n) * (1 - f_{ij})$+ $ELB(m_1..m^{cr}_{ij}, l_{i+1}..l_k) * f_{ij}$. The new schedule rating thus includes the rating from the original part of the schedule as well the ELB of the new portion of the schedule. This is basically the calculation described when the AEB was introduced in a previous section.

Now we describe the general case scenario. Let $m_1, m_2, m_3, ...m_i...m_n$ be a schedule s of n methods with k *CTER's* named $m^{cr}_{i_1 j_1}, m^{cr}_{i_2 j_2}...m^{cr}_{i_k j_k}$. Let the recovery path available at each *CTER* m^{cr}_{ij} be s^r_{ij} and each m^{cr}_{ij} occurs with frequency f^{cr}_{ij}. The AEB of the entire schedule is described recursively as $AEB = ELB(m_1...m^{cr}_{ij}, l_1, ...l_k) * f^{cr}_{ij} + AEB(m_1...\overline{m^{cr}_i}, m_{i+1}, ...m_n) * (1 - f^{cr}_{ij})$ which can be expanded out as follows:

$$\begin{aligned}
&AEB = f^{cr}_{i_1 j_1} * ELB(m_1...m_{i_1 - 1}, m^{cr}_{i_1 j_1}, l_{a1}...l_{b1}) \\
&+(1 - f^{cr}_{i_1 j_1}) * f^{cr}_{i_2 j_2} * ELB(m_1...\overline{m^{cr}_{i_1}}...m^{cr}_{i_2 j_2}, l_{a2}...l_{b2}) \\
&+ ...(1 - f^{cr}_{i_1 j_1}) * ... * (1 - f^{cr}_{i_{k-1} j_{k-1}}) * f^{cr}_{i_k j_k} * ELB(m_1...\overline{m^{cr}_{i_1 j_1}}...\overline{m^{cr}_{i_2}}...\overline{m^{cr}_{i_3}}...m^{cr}_{i_k j_k}, l_{ak}...l_{bk})+ \\
&(1 - f^{cr} i_1 j_1) * (1 - f^{cr}_{i_2 j_2}) * ... * (1 - f^{cr}_{i_k j_k}) * \underbrace{ELB(m_1...\overline{m^{cr}_{i_1}}...\overline{m^{cr}_{i_2}}...\overline{m^{cr}_{i_k}}...m_n)}_{AEUB}
\end{aligned}$$

The above computation produces an approximate measure since we use the $ELB(m_1..m_{ij}, l_{i+1}..l_k)$. A better and more exact computation would be to use the $AEB(m_1..m_{ij}, l_{i+1}..l_k)$. So if we recursively refine the $ELB(m_1..m_{ij}, l_{i+1}, ..l_k)$, the schedule rating approaches the expected bound (EB). Thus, the deeper the recursion in the analysis of *CTER's*, the better the schedule performance measure and the closer it is to the actual performance measure when rescheduling occurs. This describes the potential anytime nature of the AEB computation. Thus, in most situations, $EB(s) \geq AEB(s)$ and the $AEB(s) \geq ELB(s)$ by definition.

The optimal policy takes into account not only the robustness of the schedule being executed but also that of the contingent schedules. The AEB if modified to reschedule at critical regions and uses the AEB of the new stable schedule instead of the ELB would have a performance measure equivalent to that of the optimal policy.

Here we would like to add that all computations above are based on heuristics and hence are approximations including the OEB and EB. We could define AEUB',OEB',EB', AEB' and ELB' which would involve complete analysis of all paths by the scheduler. The resulting schedules would display higher performance characteristics and meet goal criteria better but will also be computationally infeasible to generate [37].

5.3 Rescheduling and Recovery Algorithms

In this section, we describe a generic algorithm which can guarantee a more precise performance evaluation of schedules when uncertainty is present in the schedule, using the theory described above.

Algorithm for building stable schedules:
The following is a formal description of the algorithm which chooses the schedule that provides the best performance guarantee statistically

1. Let $s^b = (m_1, m_2, m_3, ...m_i, .m_n)$ be the best schedule returned by the Design-to-Criteria scheduler for a given task structure.
2. Suppose the scheduler evaluates k schedules to decide which is the best schedule, where $s_k = (m_1^k ... m_n^k)$ and let S be the set of all k schedules.
3. s^b has the highest ELB in S.
4. Let $S_{rem} = S - s^b$. Then $ELB(s^b) \geq ELB(s)$ for all $s \in S_{rem}$.
5. Let S_{rem}^b be the set of $s \in S_{rem}$ such that $AEUB(s) \gg ELB(s^b)$. If $S_{rem}^b \neq \phi$, then we compute the $AEB(s)$ for each $s \in S_{rem}^b \bigcup s^b$.
6. The new best schedule s_{aeb}^b is the one with with the highest AEB. s_{aeb}^b is guaranteed be more robust where schedule robustness as defined earlier, is a characteristic of a schedule in which the schedule allows for recovery from execution failure of one of the scheduled actions.

Identifying CTER'S:
The AEB is a better estimate than the ELB when there is uncertainty in the schedule, i.e., there are *CTER*s in the schedule and there is a possibility for contingency plans. Earlier we defined *CTER*s as those regions in the schedule which could potentially lead to degradation in the expected performance and examined *CTER*s in the context of method failure. For example, method A2 has a quality distribution of (25% 0)(75% 3) – the 25% chance of failure makes it a candidate *CTER*. Other factors that may be used to determine whether or not a method is a *CTER* include:

1. Significant variance in the quality distribution: For methods with a single outcome, we look for variance in the quality distribution of the method with respect to the expected values and evaluate if this variance may critically affect the performance of the schedule.
2. Importance of Non-Local Effects: Certain methods may affect overall schedule performance indirectly via interactions with other tasks. For example, a given method might produce a result that has very little quality, but, a result that is needed by other *consumer* methods in the task structure. The failure of such a method may not impact overall quality directly, but, indirectly by preventing the performance of the consumer methods. Methods from which interactions originate, or from which *important* interactions originate, may also be *CTER*s.
3. Relationship between Non-Local Effects and outcomes: For methods with multiple outcomes, the variance in the quality distribution is evaluated for each outcome, as above. Additionally, any non-local-effects that are tied to particular outcomes must be examined for their importance to the overall task structure. When scheduling, each outcome has some probability of occurrence. Thus the scheduler reasons from the perspective of *all outcomes occurring* where the *likelihood of occurrence* determines the probabilities associated with non-local effects originating from particular outcomes; the uncertainty associated with the non-local effects is then propagated to the rest of the structure. To evaluate whether or not a particular method may be a *CTER* in this context requires the evaluation of each outcome and then some measurement of the probability of the outcome versus the implications of the outcome. The thresholds involved are an area of current work.

4. *Small effects*: Hereto *CTER* detection has focused on the criticality of individual methods. However, it is possible for a series of low frequency failures to be spread across several methods in such a way that no single method is a *CTER* but that the cumulative effects of the failures are equivalent to a standard, localized, *CTER*. This cumulative aggregation of *small effects* is potentially equally important as method-specific failure points because the contributing methods may be supported by recovery options as well. The OEB and EB computations in fact consider cumulative small effects of method performance because they entail rescheduling after every method execution, in the case of the OEB, and in the case of an envelope being violated in the case of the EB. The issue of what constitutes a *CTER* of this class and how to detect such *CTER*s is an area of future research.

Method reordering:

Earlier, we noted that the AEB evaluation, unlike the ELB evaluation, views permutations of the same set of methods as different schedules. We saw that while one permutation {A2,A1,A3} permitted a contingent schedule, the other {A1,A2,A3} did not. We describe below two types of method reordering within a schedule:

Simple reordering: Consider a schedule $s = \{m_1, m_2, m_3, ..m_i, ...m_n\}$. Suppose m_i is a *CTER*. Then if the AEB computation is unable to find a contingent schedule in case of failure of m_i , we will automatically try to move m_i ahead in the schedule without affecting any of the non-local effects such as enables or facilitates. So if m_i can be moved ahead of m_3 without affecting any non-local effects, we get a new schedule $s\prime = \{m_1, m_2, m_i, m_3, ...\}$ and we reevaluate the AEB rating. Our example uses simple reordering i.e. A2 can be moved ahead of A1 and a contingent schedule can be obtained. This type of reordering is always advantageous since no task interactions are lost by the reordering. The contingency analysis algorithm in this paper considers the impact of moving only one CTER ahead at a time. We plan to explore the potentially interesting albeit computationally expensive implications of reordering multiple critical regions to different levels in the future.

Complex reordering: Consider the schedule s again but suppose m_{i-1} facilitates m_i, which is a *CTER*. Also suppose we are unable to find a contingent schedule in case m_i fails. Here, we would try to move method m_i forward in the schedule, by ignoring the facilitates and evaluate if the AEB rating of the new schedule justifies the loss of the facilitates. This type of reordering is not always advantageous since the performance gain achieved by it may or may not make up for the cost involved in the detailed analysis.

Better redundancy estimation:

The relationship between the redundancy techniques employed in the main scheduler process and the recovery options explored in this secondary contingency analysis is not obvious. With respect to the redundancy techniques, contingency analysis yields better estimators of schedule performance because it factors in the probability that recovery options will be needed, and the probability that they will not be needed. In contrast, the redundancy techniques employed by the main scheduler conceptually assume either failure or success from a duration perspective, not the probability of either. Consider Figure 11 from Section 4. The standard scheduler may produce the schedule (A2, B1, B2) that contains embedded redundancy, as well as schedule (A2, B1). The schedules represent two extreme ends of the performance spectrum, one in which B1 is assumed to succeed and one in which B1 is assumed to fail. In the first case, the probability that B1 may fail is reflected in B1's expected quality and thus in the quality distribution of the schedule. However, the fact that if B1 fails, B2 must be employed, is not reflected in the quality or duration distributions of the schedule. In contrast, in the latter case, the assumption is that both B1 and B2 will be executed and the quality and duration distributions of the schedule reflect this. The extra time required to execute B2 is actually built-in to the schedule.

Regardless of whether or not B2 is actually executed, the schedule (A2, B1, B2) is evaluated on the assumption that B1 fails and B2 is required. This results in an over estimation of the time (and/or cost) that is generally required to obtain a result. In actuality, (A2, B1) will suffice 75% of the time and B2 will be required as a recovery option only 25% of the time. The exploration of this scenario via the AEUB and AEB computations correctly view these different possibilities from a probabilistic perspective and does not suffer from the over-statement problem of the main scheduler. The over estimation problem of the (A2, B1, B2) schedule is important because it may cause the main scheduler to select a different schedule for execution, i.e., it is more than a poor estimate, it may send the scheduler down the wrong path entirely. The stronger contingency analysis approach yields much better estimates and consequently leads to better decisions about which schedule(s) to execute in these cases.

An interesting extension of the evaluation in our example is to look at schedules that are produced to resolve uncertainty which in some cases instead of assuming success, assumes failure. Suppose in the Information Gathering example the results of task B is a subset of the results of task A, if task A is executed successfully. In other words the search at the Adobe site will provide only redundant information, if the Benchin site has been successfully queried. Let us assume that the new criteria is to maximize quality, a soft duration deadline of 18 minutes and a hard duration deadline of 25 minutes.

The Design-to-Criteria scheduler would then present the schedule $\{$A2,A1,A3,B$\}$ as it would have the highest ELB. So if A2 fails, execution of B would ensure that the high level goal is achieved. But the ELB computation doesn't assume rescheduling if A2 succeeds which eliminates the need to execute method B. We know $ELB(\{A2, A1, A3, B\})$ would never be better than $ELB(\{A2, A1, A3\})$ if A2 succeeded because method B is redundant and its only effect is to increase the duration of the schedule which decreases the ELB rating. In general, if the ELB criteria attaches any significance to the duration of the schedule, then the removal of actions from the schedule due to the results of prior actions making this action redundant will always increase the ELB rating.

The AEB calculation for schedules that have built-in contingencies, both successful and failure action evaluation has to be modified. Normally, contingency analysis is done for the failure region. In this case where the contingency schedule for failure is a subset of the existing schedule, one needs to do contingency analysis for both success and failure possibilities. We extend the formula described in the definition of AEB. Let $s = m_1, m_2 ... m_{ij}^{cr}, k_1, ..k_p, l_1, ..l_q, m_{i+1} ..m_n$ be a schedule s of n methods with a critical region m_{ij}^{cr} which occurs with frequency of failure f_i^{cr}. Let the recovery path available at critical region m_{ij}^{cr} be $l_1, l_2 ... l_q$ and suppose its a subset of $k_1, k_2 ..k_p$ where $k_1, k_2 ..k_p$ produces quality only if m_{ij}^{cr} succeeds and the quality produced by $l_1, l_2 ... l_q$ is independent of the success of m_{ij}^{cr}. The AEB of the entire schedule is described recursively as $AEB(s) = (1 - f_1^{cr}) * AEB(m_1, m_2 ... \overline{m_i^{cr}}, k_1, ..k_p, m_{i+1}, ..m_n) + (f_1^{cr}) * ELB(m_1, m_2 ..m_{ij}^{cr}, l_1 ..l_q, m_{i+1} ... m_n)$

So in schedule $A2, A1, A3, B$, the exact evaluation of the schedule would be one which takes both $A2^{success}$ and $A2^{failure}$ into consideration. If A2 is successful, then the methods related to failure of A2 should be eliminated (method B in this case) while rating $A2^{success}$. Likewise, if A2 fails, methods associated with the success of A2 namely A1,A3 should be eliminated while rating $A2^{failure}$. So $AEB(A2, A1, A3, B) = ELB(A2^{success} A1, A3) + ELB(A2^{failure} B)$.

6 Experimental Results

Using the measures described above, effective contingency planning is a complex process. It involves taking into account a number of factors, including task relationships, deadlines, the availability of alternatives, and client design criteria (i.e., quality, cost, duration, and certainty trade-offs). In this section, we evaluate the performance of the contingency analysis tools by comparing them to the standard Design-to-Criteria scheduler. Comparison is done by examining the Expected Lower Bound (standard scheduler metric) and the Approximate Expected Bound (contingency analysis metric) and comparing schedules selected on the basis of these metrics to the actual results obtained by executing the schedules in a simulation environment. As part of the evaluation process, we have partially determined the characteristics of task structures and design criteria that indicate a problem instance for which contingency planning is advantageous. In this section, we define the characteristics and explain why they affect performance.

The experiments in this section were conducted by randomly generating task structures while varying certain characteristics. Intuitions of which characteristics would lead to structures that are amenable to contingency analysis were used to seed the search for interesting test cases. Since method failure is a crucial factor for the contingency analysis argument, the generation of task structures was designed to concentrate on the variance of two factors, namely, the effects of failure location and failure intensity (probability of failure) within a task structure. Ten randomly generated task structure classes were then modified to varying degrees with respect to these two factors. Figure 22 shows two such randomly generated structures. In other words, ten task structure classes or prototypes were produced randomly and then these structures were modified to vary the probability of method failure and to vary the location of the method failure within all possible schedules. The latter is accomplished via non-local effects and sequencing-related quality accumulation functions that force particular actions to be carried out at particular points in any schedule including the actions.

The design criteria in these experiments is to maximize quality given a hard deadline on the overall schedule. This simple design criteria setting is one that lends itself to contingency analysis as the existence of a hard deadline (in contrast to a soft preference, e.g., soft deadline) may preclude recovery via rescheduling in certain circumstances. Because of the hard deadline, a poorly chosen initial schedule may not leave time for the deployment of recovery options and thus the normal Design-to-Criteria scheduler may fail to produce results in situations where contingency analysis has planned for the recovery scenario and chosen an initial schedule accordingly. Understanding the relationship between more interesting or diverse criteria settings and the contingency analysis is an area of current work; though results suggest that contingency analysis has benefits beyond the hard deadline (or hard cost) scenarios. For example, in some instances, contingency analysis leads to results in less time as the failure points appear earlier in the schedule.

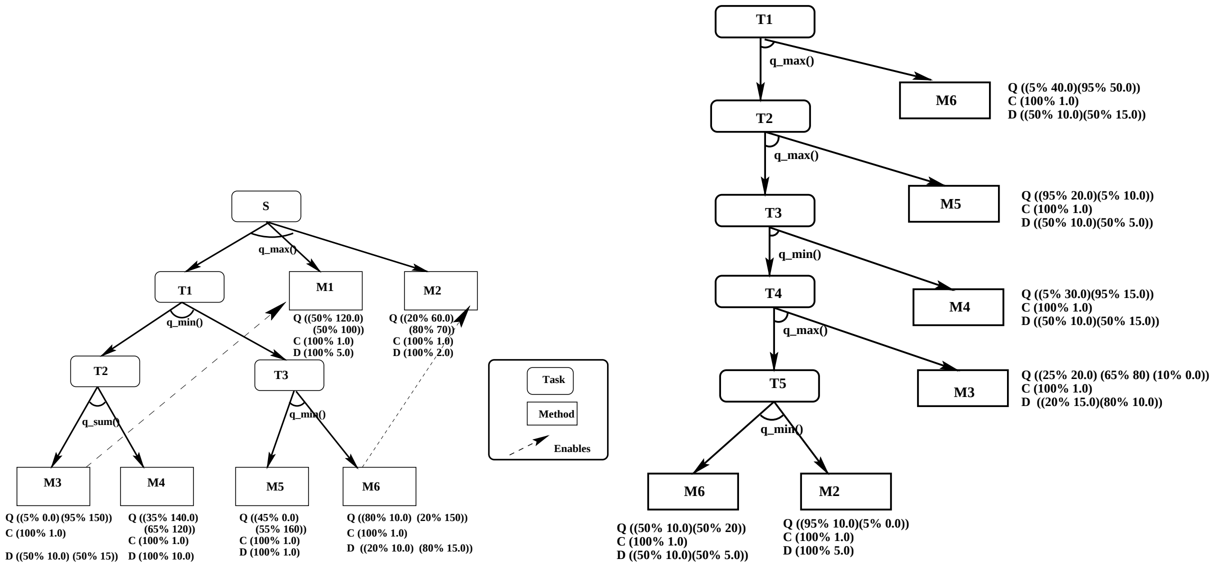


Figure 22: Sample task structures; A and B

The results for the experiments are shown in Figure 23. For each task structure instance, 100 simulated executions were performed using the schedule with the highest Expected Lower Bound(ELB) and with the schedule having the highest Approximate Expected Bound(AEB), i.e., the best schedule selected by the Design-to-Criteria scheduler was executed a 100 times and the best schedule selected by (or generated by, in the case of method movement) contingency analysis was executed 100 times. Each row in the table indicates a different *(failure location, failure probability)* parameter setting for the ten task structures; each row is also an aggregation of results for the ten task structure instances. In other words, each row represents data from an aggregate view where the ten task structure classes have been modified in a certain way to produce ten task structure instances. Of the two factors used to differentiate the task structures in each row, failure location (Lo) (found in the first column of the table) refers to the position of critical method(s) in a task structure and hence in the schedule. Failure intensity (In) (second column) refers to the probability of a method failing. Three different classifications of failure location are used in the experiments: early(E), medium(M), and late(La). Similarly, three different settings for failure intensity are used in the experiments, namely, low(L), medium(M) and high(H) where low is 1%-10% probability of failure, medium is 11%-40%, and high is 41%-90%.

For each problem instance, the execution results produced by the AEB selected schedule were compared to the results for the ELB selected schedule via statistical significance testing. The third column, *N.H. valid count*, identifies the number of problem instances for which the null hypothesis of equivalence could not be rejected at the .05 level via a one-tailed t-test. In other words, *N.H. valid count* identifies the number of experiments for which the results produced via AEB are not statistically significantly different from the results produced by the ELB. These experiments are omitted from subsequent performance measures. Generally these are instances where the schedule selected by both methodologies are the same, indicating a lack of many appealing options that may serve to lure the standard Design-to-Criteria scheduler away from the schedule that also happens to have recovery options associated with it. The elimination of many of the task structures is evidence that it is difficult to pre-determine whether contingency planning is expedient for a certain task structure.

The fourth column indicates the number of task structures of the ten possible whose data is compared. These are task structures that led to schedules for the ELB case and the AEB case that produced execution results that are statistically significantly different, i.e., the null hypothesis of equivalence was rejected at the .05 level. The remaining columns compare the AEB and ELB selected schedules' execution results for the these task structures from an aggregate perspective.

Columns five and eight, titled *Contingency A.Q* and *Normal A.Q.* respectively, show the mean, normalized quality that was produced by the AEB and ELB selected schedules respectively. In other words, the best schedule per the AEB metric was selected and executed in an unbiased simulation environment, when failure occurred the scheduler and contingency-analysis tools were reinvoked and a new schedule generated that attempted to complete the task. The resultant quality was measured and recorded and the experiment repeated 100 times. The same procedure was done for the ELB selected schedule, though when rescheduling occurred, the contingency analysis tools were not invoked (nor were they invoked in the production of the initial schedule). The overall maximum quality produced by either the AEB or the ELB simulation runs was recorded and all resultant quality then normalized over the maximum, resulting a quality value that expresses the percentage of the maximum observed quality that a given trial produced. This procedure was then repeated for the other task structure that produced statistically significantly different results, and the normalized quality values averaged. Thus, the 0.73512 A.Q. from the first row of Table 23, column four, indicates that contingency analysis yielded schedules that produced approximately 74% of the maximum observed quality on average. Column seven indicates that the standard Design-to-Criteria scheduler produced approximately 63% of the maximum observed quality, on average, for the same set of task structures. Thus, contingency analysis yielded a 14.24% percentage increase in resultant quality over the standard Design-to-Criteria scheduler, as shown in column 11.

Columns six and nine show the number of times a given selected schedule failed to produce any result within the given deadline for the AEB and ELB cases respectively. It is interesting to note that the contingency selected schedule failed to produce a result with somewhat greater frequency for rows one and five. This is because both the contingency selected schedule and its recovery option had some probability of failure, though, we do not actually consider the failure rate in these cases to be statistically significant. The failure rate in row three illustrates the classic case in which recovery before the deadline is often not possible for the schedules chosen by the standard Design-to-Criteria scheduler, whereas it is more often possible for the schedules selected by contingency analysis.

Columns seven and ten show the number of times rescheduling was necessary during execution. These results are somewhat counter intuitive as the contingency analysis selected schedules generally resulted in more rescheduling during execution due to failure. This is because the contingency analysis tools explore the possibility of recovery and do not seek to avoid the failure in the first place. Relatedly, because the contingency analysis considers the existence of recovery options, it may actually select a schedule more prone to initial failure than the standard Design-to-Criteria scheduler because the schedule has a higher potential quality. For example, say two schedules s_1 and s_2 have the following respective quality distributions: $q_1 = (25\%\ 0)(75\%\ 10)$ and $q_2 = (50\%\ 0)(50\%\ 14)$. The expected value of s_1 is 7.5 whereas the expected value of s_2 is 7. The standard scheduler will prefer s_1 over s_2 because it has a higher expected quality value (assuming that the goal is to maximize quality within a given deadline). However, the contingency analysis tools might actually prefer s_2 over s_1 if there are recovery options, e.g., s_3 for s_2, because s_2 has the potential for a higher quality result than s_1. If s_3 has a quality distribution like $q_3 = (100\%\ 7)$, then the s_2 / s_3 recovery scenario has a higher joint expected quality than does s_1 alone. Associating a cost with rescheduling in the contingency algorithms could modulate this opportunistic risk-taking type of behavior. If a cost were associated with rescheduling, the utility of a recovery option could be weighted to reflect such a cost.

The last column shows the mean normalized Optimal Expected Bound(OEB) of the AEB selected schedule. This is the measure where rescheduling is invoked after every method execution irrespective of the execution outcome. It describes the optimal performance of a schedule since the best possible path is selected every step of the way. The quality value shown is the average of 100 executions of the OEB schedule, normalized by the maximum observed quality over all the AEB selected and ELB selected schedules' executions. The OEB is higher than both *Contingency A.Q.* as well as *Normal A.Q.* for each class of task structures. This is as it should be, as the OEB is a computationally intensive performance measure which strives to obtain the optimal schedule at every point of the plan.

	Fail		N.H valid	T.S.	Contingency			Normal			Perf.	OEB
	Lo	In	count	count	A.Q.	F.R.	R.C.	A.Q.	F.R.	R.C	Impr.	
	E	M	8	2	0.73512	0/200	72	0.63041	0/200	0	14.24%	0.75227
	M	M	8	2	0.70125	2/200	64	0.63883	0/200	0	8.89%	0.71222
	La	M	8	2	0.79936	21/200	100	0.66246	38/200	48	17.12%	0.84531
	M	L	10	0	0	0	0	0	0	0	0%	0
	M	M	8	2	0.70125	3/200	64	0.63883	0/200	0	8.89%	0.71222
	M	H	10	0	0	0	0	0	0	0	0%	0
Col. #	1	2	3	4	5	6	7	8	9	10	11	12

Figure 23: *Fail Lo* is the failure location; *Fail In* is failure intensity; *N.H. valid count* is number of task structures that fail to produce results for the contingency and standard scheduler cases that are statistically significantly different; *T.S. count* is number of task structures whose performance qualities will be compared; *Contingency A.Q.* is average, normalized quality of AEB selected schedule; *Contingency F.R.* is the failure rate is number of times AEB selected schedule fails to achieve any quality; *Contingency R.C.* is the reschedule count which is the number of times the AEB selected schedule reschedules due to failure of a method to achieve quality. *Normal A.Q.* is average, normalized quality of ELB selected schedule; *Normal F.R.* is the number of times ELB selected schedule fails to achieve any quality; *Normal R.C.* is the number of times the ELB selected schedule reschedules due to failure of a method to achieve quality. *Perf. Impr* is the average improvement in performance of contingency analysis over normal scheduling. *OEB* is the average, normalized quality of AEB selected schedule.

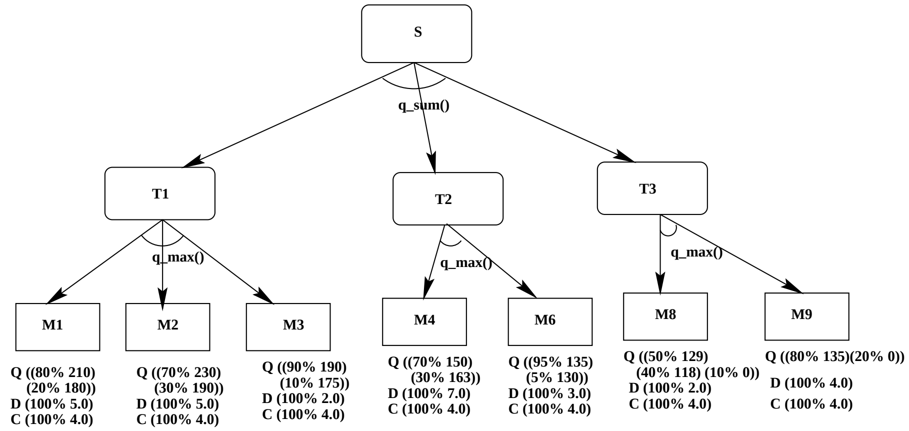
Irrespective of rescheduling, in general, for the task structures that lead to statistically significantly different results, contingency analysis produced schedules that yielded higher average quality than did the standard Design-to-Criteria scheduler. However, as illustrated by the large number of task structures that lead to results that were not statistically significantly different, very few of the candidate task structures were suitable for contingency analysis (about 20%).

Let us now step back from the aggregate view and compare contingency analysis to the standard Design-to-Criteria scheduler from a detailed perspective. Figure 24 shows a TÆMS task model on which both the standard scheduler and the contingency analysis tools were used. The expected and actual performance of the schedules produced by contingency analysis and normal scheduling techniques are described in Table 25. The design criteria is again to maximize quality within a hard deadline of 36 minutes.

The schedule selected by the contingency tools, based on the AEB, is $\{M9, M2, M4\}$ which has an ELB of 472.94, an Approximate Expected Upper Bound(AEUB) of 506.9, and an AEB of 494.21. The *CTER* in this schedule is $M9$ because $M9$ has a 20% probability of failure. Because the top-level quality accumulation function is a *sum()*, and because there are no task interactions, the failure of $M9$ is localized entirely at $M9$. This also means that a failure of $M9$, or for that matter the failure of any individual method within a schedule, will not preclude achieving some quality at the top-level task S. The contingent schedule is $\{M1, M6, M8\}$, where $M8$ is the recovery option for method $M9$. The two schedules considered by the contingency tools are hence $\{M9^{success}, M2, M4\}$ and $\{M9^{failure}, M1, M6, M8\}$.

The schedule selected by the standard scheduler, based on the ELB, is $\{M2, M4, M8\}$ which has an ELB of 484.2. The schedule is processed by the contingency analysis tools only to compute the contingency related metrics so that the schedules may be compared. The AEUB of the schedule is 494.72 and its AEB is 474.89. During the contingency analysis of this schedule, the *"move CTER forward"* heuristic moved $M8$ forward to pull the critical region closer to the front of the schedule to leave more time for recovery. Thus, the scenarios considered when computing the metrics are: $\{M8^{success}, M2, M4\}$ and $\{M8^{failure}, M1, M6, M9\}$. Regardless of the results of this analysis, the original schedule produced by the scheduler, and selected on the basis of the ELB, namely $\{M2, M4, M8\}$, is the schedule subsequently executed.

The quality achieved by the contingency selected schedule, that having the highest AEB $\{M2, M4, M9\}$, after 100 simulation runs is 502.5 which is higher than the 494.7 achieved by the best ELB schedule $\{M2, M4, M8\}$. Because $\{M9, M2, M4\}$ has a higher probability of failure, the schedule failed over twice as often as did the schedule



Sample task structure-3.

Figure 24: Task structure C

selected on the basis of its ELB. This risk-taking behavior is again because contingency analysis revealed the existence of a good quality recovery option for $M9$, namely $M8$, and that sufficient time existed to recover from a failure of $M9$. Thus, the best schedule from a quality perspective is one that includes the riskier $M9$ but also considers $M8$ in the failure case as a backup. In comparison, the standard scheduler does not consider the existence of recovery options and thus it made its choice based on expected quality alone. It is interesting to note that the ELB performance estimate for both schedules is below that which actually resulted from execution and recovery. This is related to the risk-taking behavior of the contingency analysis tools – the standard scheduler does not consider the existence of recovery options nor their value to the selected schedule. Thus the fact that when $M8$ fails, there is actually a probability of obtaining even a higher quality result by recovering and employing $M9$ is completely lost on the scheduler and not reflected in the ELB computation. This example illustrates the difference between the statistical, but local or single-schedule view employed by the Design-to-Criteria scheduler and the more accurate, contextual view, generated by performing contingency analysis on the schedules produced by the scheduler.

Schedule Analysis	Schedule Produced	ELB	AEUB	AEB	Resched count	Actual Quality
Contingency	$\{M9, M2, M4\}$	472.94	506.9	494.21	23	502.35
Normal	$\{M2, M4, M8\}$	484.2	494.72	474.89	12	495.13

Figure 25: Performance information for task structure C

We now evaluate the performance of the contingency analysis algorithm by comparing it to the performance of the optimal policy produced by the MDP-based meta-control system. The experiments in this section were performed on the same 8 task structures whose rating of schedule with highest Approximate Expected Bound(AEB) were found to be statistically significantly higher than the schedule with the highest Expected Lower Bound(ELB). The design criteria is to maximize quality within a given deadline.

The results of our experiments are shown in Figure 26. For each of the task structures, we computed the average quality achieved by the schedule with highest ELB, schedule with highest AEB and the optimal policy over 100 simulations. Rescheduling in the event of failure within the deadline is permitted. For each task structure, the qualities achieved in all three cases were normalized by the actual ELB estimate[19] of the highest rated ELB schedule for that

[19] this is the statistical measure computed by DTC's rating mechanism and it assumes no rescheduling. This is in contrast the actual qualities achieved by the schedule with highest ELB during simulations, which allow for rescheduling in the event of failure.

task structure.

The histogram on the left shows the normalized average quality of the schedules with the highest ELB for each task structure. In six of the eight cases, the estimated ELB (with no rescheduling) is equivalent to the actual ELB, since there were no reschedulings in those six cases. There is a slight improvement in performance in the 2 cases when there is rescheduling. However, this observation cannot lead to the generalization that rescheduling leads to improved performance. The performance of the schedule with recovery from failure does not necessarily have to be better than that of the original schedule with no no failure. Also if there is a cost associated with rescheduling, the option with failure recovery and higher quality becomes less desirable than the option with no failure and lower accrued quality. The histogram in the middle shows the normalized average quality of the schedules with highest AEB for each task structure, with rescheduling in the event of failure. In all eight cases, the quality is higher than the estimated ELB as well as the actual average ELB value. The histogram on the right shows the normalized average quality of the optimal policy and in all eight cases it is better than the DTC scheduler (with and without rescheduling).

For each task structure, it can be observed the Performance(optimal controller) $\geq$ Performance(contingency enhanced scheduler) $\gg$ Performance(normal scheduler with rescheduling on failure) $\geq$ Performance(normal scheduler with no rescheduling on failure).

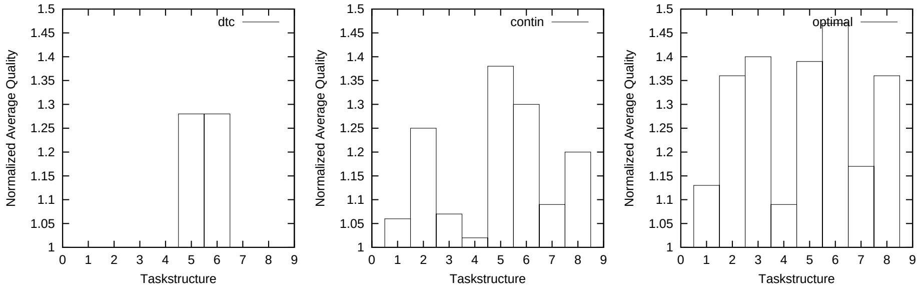


Figure 26: Performance of DTC, contingency enhanced DTC and Optimal Selected Schedules relative to estimated highest ELB for the task structure

The above experimental results leads to the following performance characterization of the various schedulers for a certain class of task structures/problems which exhibit properties of mission-critical systems [26] i.e., they have critical task execution regions and are constrained by hard deadlines and mid-stream schedule failure could lead to catastrophic system-wide failure.

1. The performance of the contingency-enhanced DTC scheduler is significantly greater than the standard DTC scheduler.
2. The performance of the contingency-enhanced DTC scheduler is in some cases equivalent to the performance of the optimal policy.
3. It is advantageous to reschedule in the event of failure as long as the overhead associated to rescheduling is minimal.

Based on the results presented here and other similar results, it is possible to characterize the types of task structures that are amenable to contingency analysis, i.e., those for which analysis of recovery options is beneficial from a cost/benefit perspective. The general characteristics include:

1. Methods in task structures should have a possibility of failure in their distribution. Contingency analysis is worth the associated computational overhead only if there is a possibility of failure of the current schedule to meet the high-level goal due to individual method failure. If the performance of the best schedule is deterministic, contingency analysis is dispensable.
2. Task structures should contain alternate paths. The absence of of possible recovery paths in the face of failure also makes contingency analysis dispensable.

3. Task structures should contain alternate paths with some overlapping structure(preferably in the initial stage) and with significant performance differences. For instance, suppose a task structure has a path {A,B,C,D} with high potential quality but also a high risk of failure, and another path to reach the same high-level goal, namely {A,B,E,F,G}, has low quality, low duration but has no possibility of failure. The presence of an alternate path that achieves some quality and takes advantage of the successful work already done is akin to the conceptual notion of a "quick and dirty" approach to problem solving. The existence of these types of methods provide the contingency planning approach with a recovery option that is usable even in tight resource situations.
4. A possibility of moving failure methods forward (absence of associated hard non-local effects) would further the potential of contingency analysis, i.e., structures in which there is some flexibility in terms of method placement within a schedule. If methods have strong precedence and succession constraints by way of enables non-local effects, and the failure points are in the latter portion of the schedule, then there is little possibility of finding good recovery options for failure within the resource constraints.
5. Dependence of methods with good average performance on critical methods (enables non-local effect from a critical method to a non-critical method). Extensive contingency analysis is required only if the critical regions affect the rest of the schedule significantly. Otherwise, a cheap local fix by replacing the critical method by a more stable method or just adding a redundant method within the criteria requirements is a better choice than contingency analysis.

The following are the characteristics of the design criteria which augments contingency planning.

1. The objective function could specify a hard deadline, and emphasis should be given to either the quality or duration slider. The hard deadline and other such hard resource constraints voids the possibility of simply rescheduling at failure points and instead requires off-line contingency analysis.
2. The deadline should also provide enough time for contingency analysis, if the scheduling cost is factored into the equation. Regardless, the deadline must provide sufficient time for recovery options to be deployed otherwise the existence of such options is meaningless. In these cases, the contingency analysis tools must resort to the same single-pass execution view that is used in the main Design-to-Criteria scheduler.

7 Conclusions and The Future Role of Uncertainty

Dealing with uncertainty as a first class object both within the scheduling process and via the secondary contingency analysis is beneficial. The addition of uncertainty to the TÆMS modeling framework increases the accuracy of TÆMS models. The uncertainty enhancement is leveraged ubiquitously by Design-to-Criteria scheduling to reason, from a probabilistic perspective, about the performance characteristics of primitive actions and task interactions. Including explicit models of uncertainty improves the scheduling process not simply by increasing modeling power, but also by increasing the representational power of all the computations in the scheduling process. As discussed in Section 4.3, the probabilistic models occasionally adversely affect the scheduler calculations, but, even with the loss of the independence assumption, improvement of computation accuracy outweighs the associated computation costs.

As discussed in Sections 4.1 and 4.2, integration of uncertainty in the client goal or design criteria specification enables clients to describe the relative importance of certainty, and uncertainty reduction, to a particular application. Integration of this metric into the utility calculations that govern scheduler problem solving enables the scheduler to evaluate quality, cost, duration and quality-certainty, cost-certainty, and duration- certainty trade-offs of particular courses of action. This integration approach enables clients to specify the balance between uncertainty reduction and the other utility metrics, i.e., uncertainty reduction does not dominate the problem solving process *unless* so specified by the client. The integration and use of uncertainty in the main Design-to-Criteria scheduling process provides a means for reasoning about, and working to reduce, uncertainty within the confines of addressing soft real-time scheduling deadlines and other real performance constraints.

The secondary contingency analysis procedures presented in Section 5 step outside of this context to perform a more detailed analysis of schedule performance based on the existence of recovery options. Since the algorithms explore the schedule recovery space using the Design-to-Criteria scheduler, they still exhibit a satisficing, approximate, resource conservative nature. It is interesting to note that even the coarse analysis performed in the Approximate Expected Bound (AEB) and Approximate Expected Upper Bound (AEUB) computations is beneficial in certain circumstances. Future efforts in contingency analysis will involve explicitly bounding and controlling the complexity of the contingency analysis process. Intertwined with this research objective is the ability to classify particular problem solving instances. From the experiments performed in Section 6, it is clear that certain classes of task structures are more amenable to contingency analysis than others. Contributing factors include the location of the failure point and the number and quality of recovery options available. These task structures exhibit mission-critical properties i.e. they have critical task execution regions and are constrained by hard deadlines and mid-stream schedule failure could lead to catastrophic system-wide failure. Given the ability to classify task structures, an input task structure could be examined to determine 1) whether or not contingency analysis should be performed and 2) if the analysis should be performed, how deep the algorithms should search when exploring recovery options. In some cases, deep exploration may not be fruitful and in others, it may be critical.

Section 6 that the contingency-enhanced Design-to-Criteria scheduler performs pretty good for a certain class of task structures,

Another area of future exploration in contingency analysis lies in the determination of critical regions, *critical task execution region*s (*CTER*s), within schedules. One aspect of this is determining *CTER* status based on the existence and types of task interactions. Another aspect is in the determination of *CTER* status by examining the cumulative or aggregation of low frequency failures in methods. The algorithms discussed earlier focus on a *local* determination of criticality, that is, as being localized in a given method. However, it is possible that low frequency failures spread across multiple methods may also result in a critical region within a given schedule. This *small effects* condition may also benefit from the existence of recovery analysis and contingency planning. Another related area is that of dynamically re-evaluating the *CTER* status of methods. In this work, we considered only static critical task execution regions i.e. the identification of critical task execution regions is independent of the progressive results of schedule execution. However, as execution unfolds, methods that are not critical to begin with may become more important. In general, changing context is handled by rescheduling, however, envelopes or triggers could be specified and examined incrementally during execution, akin to [1].

Related to the issue of envelopes is caching the recovery options explored and identified during contingency analysis. As the recovery options are explored from a statistical perspective, where primitive actions have ranges of characteristics, it is not immediately clear that storing the recovery options and deploying them automatically in the case of failure is a good solution. This is somewhat related to the issue of *small effects* discussed earlier in that during actual execution, values are produced and while a single value may not fall outside of a conventionally generated rescheduling envelope (e.g., reschedule if results are not within 25% of the trimmed mean), the cumulate effects of the results may lead to different recovery options being more desirable in the event of a failure. Because the scheduler and contingency analysis tools reason about non-local effects from a probabilistic perspective, these aggregation effects may be even more pronounced than the *small effects* dealt with in *CTER* determination (as the *CTER* computation uses the same probabilistic view used in the rest of the scheduler computations).

In the current implementation, the cost of rescheduling is considered to be a small overhead and the costs associated with rescheduling are effectually ignored. This has proven effective in practice and schedule times tend to be less than one percent of total system execution time. However, we are currently exploring the implications of scheduling in environments where the rescheduling cost is non-trivial. In such situations, interesting techniques like including slack time in schedules or viewing a schedule as partially-ordered set of methods (rather than a linearly ordered set of methods) may be appropriate.

Another area of future uncertainty-related work in Design-to-Criteria scheduling involves leveraging the uncertainty-enhanced TÆMS models in multi-agent scheduling and coordination. In multi-agent systems the scheduler is typically coupled with a multi-agent coordination module that forms commitments to perform work with other agents; local concerns are thus modulated by non-local problem solving. Uncertainty in this context could be used to reason about the utility of the commitments made with other agents and to understand how the uncertainty about commitments made by other agents affects local problem solving [40].

We also plan to look into more efficient methods for determining the optimal policy in the MDP framework. Previous research in the area has shown that the stochastic nature of some MDPs has important consequences for complexity and not all MDPs are equally difficult to solve. Classification of task structures to help pre-determine the complexity of the MDP would be useful in deciding whether to use the heuristic scheduler or MDP-based optimal controller for specific problems. We also plan to evaluate policy performance by applying existing approximation techniques (fast ϵ-approximation algorithms and Markovian chains) which trade-off solution accuracy for time.

8 Acknowledgments

We would like to acknowledge the work and contributions of Professor Alan Garvey. His Design-to-Time scheduling work provided the conceptual foundation for subsequent work in this area. We also thank Professor Shlomo Zilberstein for his contribution in the presentation of the contingency analysis algorithm as well as the discussions on the translation of TÆMS task structures to Markov Decision Processes.

References

[1] R. St. Amant, Y. Kuwata, and P. Cohen. Monitoring progress with dynamic programming envelopes. In *Proceedings of the Seventh International IEEE Conference on Tools with Artificial Intelligence*, pages 426–433, 1995.

[2] A.G. Barto, S. J. Bradtke, and S.P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72:81–138, 1995.

[3] A. L.C. Bazzan, V. Lesser, and P. Xuan. Adapting an Organization Design through Domain-Independent Diagnosis. Computer Science Technical Report TR-98-014, University of Massachusetts at Amherst, February 1998.

[4] C. Boutilier, T. Dean, and S. Hanks. Planning under uncertainty: Structural assumptions and computational leverage. In *Proceedings of 3rd European Workshop on Planning (EWSP'95)*, 1995.

[5] J. Bresina, M. Drummond, and K. Swanson. Just-in-case scheduling. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 1994.

[6] N. Carver and V. 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.

[7] T. Dean and M. Boddy. An analysis of time-dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 49–54, St. Paul, Minnesota, August 1988.

[8] T. Dean, L. Kaelbling, J. Kirman, and A. Nicholson. Planning under time constraints in stochastic domains. *Artificial Intelligence*, 76(1-2):35–74, 1995.

[9] R. Dearden and C. Boutilier. Abstraction and approximate decision-theoretic planning. *Artificial Intelligence*, 89:219–283, 1997.

[10] K. Decker and Jinjiang Li. Coordinated hospital patient scheduling. In *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS98)*, pages 104–111, 1998.

[11] K. S. Decker and V. R. Lesser. Quantitative modeling of complex environments. *International Journal of Intelligent Systems in Accounting, Finance, and Management*, 2(4):215–234, December 1993. Special issue on "Mathematical and Computational Models of Organizations: Models and Characteristics of Agent Behavior".

[12] K. S. Decker and V. 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.

[13] D. Draper, S. Hanks, and D. Weld. Probabilistic planning with information gathering and contingent execution. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems (AIPS-94)*, pages 31–36, 1994.

[14] A. Garvey, M. Humphrey, and V. Lesser. Task interdependencies in design-to-time real-time scheduling. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 580–585, Washington, D.C., July 1993.

[15] A. Garvey and V. Lesser. Design-to-time real-time scheduling. *IEEE Transactions on Systems, Man and Cybernetics*, 23(6):1491–1502, 1993.

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

[17] A. Garvey. *Design-to-Time Real-Time Scheduling*. PhD thesis, University of Massachusetts at Amherst, Amherst, Massachusetts, February 1996.

[18] P. Haddaway and S. Hanks. Utility models for goal-directed decision-theoretic planners. *Computer Intelligence*, 14(3), 1998.

[19] E. Horvitz, G. Cooper, and D. Heckerman. Reflection and action under scarce resources: Theoretical principles and empirical study. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, August 1989.

[20] E. Horvitz and J. Lengyel. Flexible Rendering of 3D Graphics Under Varying Resources: Issues and Directions. In *Proceedings of the AAAI Symposium on Flexible Computation in Intelligent Systems*, Cambridge, Massachusetts, November 1996.

[21] D. Jensen, M. Atighetchi, R. Vincent, and V. Lesser. Learning Quantitative Knowledge for Multiagent Coordination. *Proceedings of AAAI-99*, 1999. Also as UMASS CS Technical Report TR-99-04.

[22] S. Sutton Jr. and L. Osterweil. The design of a next-generation process language. In *Proceedings of the Fifth ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 142–158, September 1997.

[23] N. Kushmerick, S. Hanks, and D. Weld. An algorithm for probabilistic planning. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 1994.

[24] V. Lesser, M. Atighetchi, B. Horling, B. Benyo, A. Raja, R. Vincent, T. Wagner, P. Xuan, and S. XQ. Zhang. A Multi-Agent System for Intelligent Environment Control. In *Proceedings of the Third International Conference on Autonomous Agents (Agents99)*, 1999.

[25] V. Lesser, B. Horling, F. Klassner, A. Raja, T. Wagner, and S. XQ. Zhang. BIG: An agent for resource-bounded information gathering and decision making. *To appear in the AIJ*, 2000.

[26] D. J. Musliner. Plan Execution in Mission-Critical Domains. In *Working Notes of the AAAI Fall Symposium on Plan Execution - Problems and Issues*, 1996.

[27] T. Oates, M. V. Nagendra Prasad, and V. R. Lesser. Cooperative Information Gathering: A Distributed Problem Solving Approach. Computer Science Technical Report 94–66, University of Massachusetts, 1994. Journal of Software Engineering, Special Issue on Developing Agent Based Systems, 1997.

[28] N. Onder and M. Pollack. Contingency selection in plan generation. In *Proceedings of the Fourth European Conference on Planning*, 1997.

[29] S. J. Russell and S. Zilberstein. Composing real-time systems. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pages 212–217, Sydney, Australia, August 1991.

[30] H. A. Simon. *Administrative Behavior*. Macmillan Company, New York, NY, 1945.

[31] H. A. Simon. *Models of Bounded Rationality*. MIT Press, Cambridge, MA, 1982.

[32] W. Slany. Scheduling as a fuzzy multiple criteria optimization problem. *Fuzzy Sets and Systems*, 78:197–222, March 1996. Issue 2. Special Issue on Fuzzy Multiple Criteria Decision Making; URL: ftp://ftp.dbai.tuwien.ac.at/pub/papers/slany/fss96.ps.gz.

[33] J. Tash and S. Russell. Control strategies for a stochastic planner. In *Proceedings of the Eleventh National Conference on Artificial Intelligence,*, pages 1079–1085, 1994.

[34] T. Wagner. *Toward Quantified Control for Organizationally Situated Agents*. PhD thesis, University of Massachusetts at Amherst, Amherst, Massachusetts, February 2000.

[35] R. Vincent, B. Horling, T. Wagner, and V. Lesser. Survivability simulator for multi-agent adaptive coordination. In *Proceedings of the First International Conference on Web-Based Modeling and Simulation*, 1998.

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

[37] T. Wagner, A. Garvey, and V. Lesser. Criteria-Directed Heuristic Task Scheduling. *International Journal of Approximate Reasoning, Special Issue on Scheduling*, 19(1-2):91–118, 1998. A version also available as UMASS CS TR-97-59.

[38] T. Wagner and V. Lesser. Toward Ubiquitous Satisficing Agent Control. In *1998 AAAI Symposium on Satisficing Models*, March, 1998.

[39] T. Wagner and V. Lesser. Relating quantified motivations for organizationally situated agents. In N.R. Jennings and Y. Lespérance, editors, *Intelligent Agents VI — Proceedings of the Sixth International Workshop on Agent Theories, Architectures, and Languages (ATAL-99)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, Berlin, 2000.

[40] P. Xuan and V. R. Lesser. Incorporating uncertainty in agent commitments. In N.R. Jennings and Y. Lespérance, editors, *Intelligent Agents VI — Proceedings of the Sixth International Workshop on Agent Theories, Architectures, and Languages (ATAL-99)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, Berlin, 2000.

[41] S. Zilberstein and S. J. Russell. Optimal composition of real-time systems. *Artificial Intelligence*, 82(1):181–214, December 1996.

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

[43] S. Zilberstein and S. J. Russell. Constructing utility-driven real-time systems using anytime algorithms. In *Proceedings of the IEEE Workshop on Imprecise and Approximate Computation*, pages 6–10, Phoenix, AZ, December 1992.

[44] M. Zweben, B. Daun, E. Davis, and M. Deale. Scheduling and rescheduling with iterative repair. In M. Zweben and M. Fox, editors, *Intelligent Scheduling*, chapter 8. Morgan Kaufmann, 1994.