DRAFT: Multi-Agent Decision Support Via User-Modeling

Terrence Harvey, Keith Decker, and Sandra Carberry
University of Delaware
Computer and Information Sciences
{harvey, decker, carberry}@cis.udel.edu

ABSTRACT

Decision-support requires the gathering and presentation of information, but is subject to many kinds of resource restrictions (e.g. cost, length, time). Individual users differ not only in the resources they have available to expend, but also in the priorities they place on different kinds of information. While it is straightforward to represent these differing priorities and related constraints in a user model, using that model to allocate resources for an unseen task across multiple agents in a dynamic environment is not as simple. Before the information gathering process begins, it is not known which agents will be able to usefully participate, or how much utility they will ultimately be able to provide.

MADSUM is a distributed adaptive system that uses a negotiation process to solicit and organize agents to produce information, and a presentation assembly process to coherently assemble the information into text for decision support. MADSUM assumes poor predictive models of ultimate information utility and thus requires dynamic organizational management in response to run-time information failures.

A user model, including content preferences, deadlines, and length constraints, informs both processes. An evaluation demonstrates that the influence of the user model on content selection and presentation improves system output, and that the organization responds appropriately and predictably in the presence of inevitable information failures.

Categories and Subject Descriptors

H.5.2 [User Interfaces]:Natural Language
I.2.11 [Distributed Artificial Intelligence]:Multiagent Systems

General Terms

Algorithms

Keywords

User modeling, robustness, decision support

AAMAS'05, July 25-29, 2005, Utrecht, Netherlands.

1. MOTIVATION

An effective decision support system must produce valued information while taking into account the user's constraints on resources, the user's priorities on resources, and the user's priorities on kinds of information that he is most interested in receiving. Many of these constraints and priorities directly affect the manner in which the system can achieve its goals. For example, a user's limits on cost and time translate directly into constraints for the system.

However, the user's priorities will change over time, requiring a system that can manage varying constraints and preferences. Furthermore, a system operating in a dynamic information environment must implement these changing considerations while the world changes, too.

These observations motivated the development of MADSUM (Multi-Agent Decision Support Via User Modeling), a hierarchical agent system whose choices are driven by a user model containing information about the user, the user's resource constraints, and the user's priorities. MADSUM has two primary contributions. First, it develops text plans for decision support while balancing, in a decision-theoretic manner, the relative values to a user of multiple resource and information attributes. Second, MADSUM's implementation relies on an agent failure-handling protocol that facilitates flexible, fail-soft performance in a dynamic environment.

MADSUM is introduced in Section 2. Section 3 presents the MADSUM architecture and its approach to dynamic response generation, with an emphasis on resource allocation. Section 4 gives an example of how MADSUM balances resource and content priorities along with information significance to produce adaptive responses. Our approach to result failure management is described in Section 4.1. Section 5 presents evaluation experiments that demonstrate both MADSUM's success at adaptive response generation and its promising time trials. Sections 6 and 7 then discuss related work and our conclusions.

2. DISTRIBUTED DECISION SUPPORT

We have been investigating the design of a decision support system that can adapt to a user's resource constraints, resource priorities, and content priorities in a dynamic environment. Our approach consists of a hierarchy of cooperative agents that consider the preferences and constraints of a user while gathering and assembling information.

Agent decisions are guided by a multi-attribute utility function as part of a user model. The utility function weighs the benefit of different decisions about resource usage and information selection. The user of the system can tailor the utility function to affect: the information content of the result (which kinds of information are included); attributes of the resulting message itself (such as length

and cost); and attributes of the planning process (e.g. time). Our approach provides a structure in which the priorities of the user can be explicitly represented and considered in light of the environment (information currently available, the cost of getting the information, etc.). Furthermore, the use of an appropriately designed agent architecture allows the system to dynamically respond to changes in the environment and/or user priorities.

We have applied the MADSUM architecture to decision-support in a financial investment domain, where the system must support a user in making a buy/don't-buy decision on a single investment. The MADSUM decision making algorithms and the agent hierarchy, communications, and interaction are domain-independent. Furthermore, MADSUM is easily extended to new domains by adding different attributes to its utility function. However, implementation in a particular domain requires a set of domain-dependent information agents. These agents must "wrap" any external information sources, but also must provide limited analysis. For example, tailored decision-support in an investment domain requires domain-dependent agents that can estimate how significant a particular piece of information will be to the current user, given her stated priorities and current personal and financial status.

3. AGENT ARCHITECTURE

Information of all kinds is increasingly available from distributed sources. Multi-agent systems are used as a means of addressing the problems and opportunities presented by a dynamic, heterogeneous information environment[10]. The DECAF architecture was selected for its built-in support for communication between agents running on different systems, its graphical agent design interface, and its ability to make scheduler choices at runtime (a feature which MADSUM does not currently leverage, but will use in future work).

3.1 The DECAF Agent Internal Architecture

DECAF (Distributed, Environment-Centered Agent Framework) is an agent architecture and toolkit for quickly prototyping multi-agent information gathering systems that can represent and manage the complexities and uncertainties of distributed information[7]. The toolkit provides a stable platform to design, rapidly develop, and execute intelligent agents to achieve solutions in complex software systems. DECAF provides the necessary architectural services of a large-grained intelligent agent [5, 16]: communication, planning, scheduling, execution monitoring, coordination, and eventually learning and self-diagnosis.

DECAF provides an environment that allows the basic building block of agent programming to be an agent action, or a pre-specified subtask (collection of agent actions). These building blocks are then chained together by the DECAF planner. This paradigm differs from most of the well-known agent toolkits, which instead use the API approach to agent construction (e.g., [15]). Functionally, DECAF is based on RETSINA [16] and TÆMS [4].

Figure 1 represents the high level structure of a single DECAF agent. Structures inside the heavy black line are internal to the agent architecture and the items outside the line are user-written or provided from some other outside source (such as messages from other agents).

As shown in Figure 1, there are five internal execution modules (square boxes) in the current implementation, and seven associated data structure queues (rounded boxes). DECAF is multi-threaded, and thus *all modules execute concurrently, and continuously* (except for agent initialization).

3.2 MADSUM Architecture

To address the issues of collecting and integrating information from distributed sources into a single text plan, MADSUM is implemented as a hierarchical network of independent agents. Though designed for an arbitrary number of task and source agents, the current implementation consists of thirteen DECAF software agents. The Presentation Agent is at the top of the hierarchy, and is the agent that receives requests from a user and delivers the final decision support text. At the lowest level are seven information providers that can access information, sometimes from remote sources. The internal task agents of the hierarchy each have the capacity to make independent decisions about which information to pass upward from their children and how to recover when results from agents below fail to meet expectations. This distributed structure allows quick development, incorporation, and maintenance of source "wrappers" and agents with expertise in different areas; the incorporation of agents managed by other organizations; rapid movement in and out of the system by agents; and some benefits of parallelization and fail-soft behavior due to process distribution[8].

MADSUM's final product, a text designed for decision support, is the result of a negotiation process consisting of four parts based on the FIPA contract net protocol[6]. First, a decision-support task, a utility function, and a set of soft and hard constraints are passed to all agents. Next, agents bid by submitting multiple estimates for results they expect to be able to provide, expressed as a series of attributes. Third, an agent commits to specific bids from its children; and finally gathering, integration, and propagation of results occurs. At each point decisions are made based on the utility function and other aspects of the user model.

3.2.1 The User Model

MADSUM's user model has three components: User Attributes, a Utility Function, and a set of hard and soft constraints (hard constraints on an attribute are limits beyond which overall utility becomes zero, whereas exceeding a soft constraint only diminishes utility). Although User Attributes might be captured in a long-term user model that is constructed and revised over time, the constraints and the Utility Function will vary with different user interactions and perhaps even change during an interaction.

The User Attributes component of the user model captures characteristics of the user, including appropriate domain-specific information. For the financial investment domain, this component of the user model includes the user's age, salary, expected number of years to retirement, approximate annual expenditures, current investment portfolio, and portfolio allocation goals. The user attributes affect the calculated significance of certain pieces of information. For example, the more a proposed investment would cause one's investment portfolio to deviate from one's portfolio allocation goals, the more significant is information about the deviation.

3.2.2 Utility Function

The intent of every decision support system is to be effective, but effectiveness is in the eye of the beholder. Specifically, the needs of the user in the context of a particular environment determine whether certain information will be deemed supportive or of little worth. Identifying these needs is crucial to the system's ability to satisfy the user, but is subject to imperfect methods (e.g. asking or observing the user) and assumptions (e.g. a rational user). Other work focuses on this aspect of the decision support problem (see [1, 13]). MADSUM asks the user for relative preferences about information topics, to be input via graphical sliders, and has default settings for other attributes which an advanced user can alter. Given these preferences we establish a function for calculating user

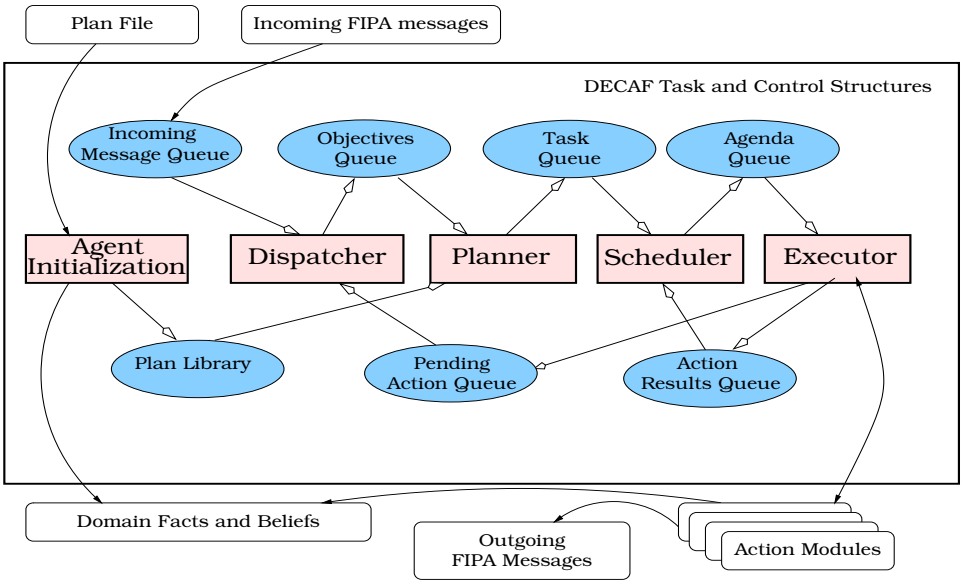

Figure 1: DECAF Architecture Overview

utility. Utility is a number used during planning to approximate effectiveness given (necessarily) incomplete models of the user and the environment.

MADSUM's utility function contains n attribute terms, each consisting of a weight w_i giving the importance of that attribute to the user, a parameter a_{value_i} that is related to the value of the attribute, and a function f_i.

$$Utility = \sum_{i=1}^{n} w_i f_i(a_{value_i}) \quad (1)$$

The weights w_i, giving the importance of each attribute to the user, are extracted from the positions of sliders that are manipulated by the user in a graphical user interface. For resource attributes such as length of response or processing time, a_{value_i} is the actual value of the attribute, such as 75 words. On the other hand, information attributes capture propositions that might be presented to the user, and thus for information attributes, a_{value_i} captures the significance of a set of propositions to the decision task at hand — ie., its significance in the environment of the user's personal characteristics and the application domain. We call this approximation *Decision Specificity* or *DS*. Determining DS is a domain-specific task, and thus in the MADSUM architecture, the functions that compute DS are provided by the application designer as part of the domain-specific information agents that propose propositions for inclusion in the response to the user.

In the financial investment domain, we have implemented domain-specific information agents for three categories of information: Risk (the riskiness of an investment), Value (the prospects for the investment gaining in value), and Portfolio (how the investment relates to the individual's portfolio allocation goals). The associated decision specificity functions produce estimates of significance that are referred to as DS_r, DS_v, and DS_p respectively. It is important to note that DS corresponds not with the precise attribute value but instead with the significance of the information.

Each of the base utility functions f_i that appear in the user utility function map their parameter a_{value_i} into a value between 0 and 1. The particular function f_i that is used determines whether a larger parameter value increases or decreases utility (and at what rate). These functions are chosen from an extendable library of functions with varying characteristics, including variations on linear, normal, and plateau functions and combinations of thereof. The soft constraints entered by the user adapt each f_i by determining its shape. Exactly how the shape is modified depends on the f_i, but soft constraints can affect mean, spread, or direction change points.

Thus the magnitude of a term in the overall utility function is affected by the value of the attribute in the environment (either its actual value in the case of resource attributes or the DS value computed from propositions in the case of information attributes), the base utility function f_i, and two user-selected modifiers (the weight w_i that gives the importance of this attribute to the user, and the soft constraint that adapts the function f_i). Allowing the user flexibility in designing each term's contribution to utility ensures that the resulting utility function reflects not only the attribute value, but also the user's opinion of how the attribute contributes to utility.

3.3 Allocation of Resources

MADSUM assumes that it is difficult to predict the utility of an information-gathering task before the task is performed. Yet if resources such as cost and time are limited, an attempt must be made to distribute those resources across agents in a way that maximizes utility. When utility functions are static, when a user can answer a series of questions about the domain, or when the function is simply additive, it is possible to determine the degree to which each attribute of a result contributes to overall utility and allocate resources in a decision-theoretic manner. These assumptions do not apply in our domain. For example, domain-specific processing might recognize that a result from one agent will enhance (or diminish) the value of the result from another; but this cannot be determined by a generic agent component examining the separate attributes of the results.

Our approach is to allocate expected utility instead of resources. This MADSUM design decision is consistent with the DECAF strategy of committing to objectives, not to the plans that achieve them. Committing to objectives allows agents to achieve the objectives with the strategy that best utilizes the environment at runtime. For example, a contractor should be able to commit to building a house for a certain price, and by a certain date, without having to

specify where materials are going to be purchased or exactly which subcontractors will be used. Allocating expected utility makes even more sense in decision-support, where the objectives of the process and the utility of the result are almost the same thing (in contrast to building widgets, for example).

The Presentation agent always seeks maximum utility under the current utility function. After the bidding process, when a specific utility objective for this decision support task is determined, the agent sets a *utility envelope* for each child agent based on the accepted bid from that child. The purpose of an envelope is to allow some flexibility in accepting responses that are not exactly as negotiated. This strategy reflects the real costs, in agent time, computation, and communication, associated with rejecting a response and trying to find an alternative. The default envelope minimum is 95 percent of the utility offered by the accepted bid. If an agent has bid an alternative in addition to its accepted bid, then the cost of rejecting the result of the accepted bid is lowerd, since finding that alternative is trivial. Thus when an agent has an alternative bid whose utility is higher than 95 percent of the accepted bid, the alternative bid's utility becomes the minimum acceptable utility.

To clarify the envelope concept, consider a case where the only attribute of the result that is changing is the cost. For example: you have decided to buy a widget for $10.00. When you get to the cash register, you find that the widget is actually a different price. Under what circumstances would this deter you from the purchase? If the cost was very close, you might decide that purchasing the widget at a higher price was your best course. However, if the widget was now $10.50 and you had seen another widget in the same store marked $10.20, you might decide to reject the higher priced widget and walk a few feet to get the lower priced alternative. In other words, your willingness to accept something other than what you expected depends on your alternatives (as well as hard constraints such as how much cash you brought to the widget store; hard constraints are examined separately from utility envelopes).

One advantage to doing using a utility envelope is that it is simple and fast. The utility an agent agrees to provide is expected from its children in proportion to the utility of their contibutions. If the utility of the whole is greater than the sum of the utilities of the parts, then children are assigned an envelope with the utility of their original bid, and the parent agent assumes that the combined results will again exhibit gestalt properties.

Unfortunately, there is no guarantee that given only an expected utility, an agent will consume resources in the same quantity it originally proposed. However, our system consists of cooperative agents which share a common utility function, so in the usual case they can be expected to approach their predicted consumption. For the other cases, MADSUM agents monitor the results produced by their children and have an elaborate but speedy protocol (see Section 4.1) for addressing results that do not meet specifications.

One problem faced when allowing independent branches of an agent tree to make commitments is that two branches may both commit to the maximum consumption of some resource. For example, in the text planning domain every source agent could submit a bid that would produce the length of the entire desired result. This would leave agents above with the option of allowing only a single source agent to contribute, or forcing all bidders to re-bid; and in the latter case, nothing would stop the same problem from occuring again, and being repeated at each level of the tree. The MADSUM architecture addresses this issue by allowing agents to make improper allocations, but employing strategies to avoid this; and by focusing on recovering from such failures gracefully (see Section 4.1).

In summary, there are several important features of the MADSUM negotiation process:

- Agents are presented with the user's utility function and the total resources available, as well as any overall constraints, so that they can contribute to the best of their ability in all cases.
- Agents can present more than one bid, so that they can offer options for a range of utility/cost tradeoffs. Agents present not just high utility options, but options that differ widely in parameter space, to allow maximum flexibility should the utility function change mid-process.
- A utility envelope allows bidders some flexibility in the utility they ultimately provide, but that flexibility is partially dependent on the utility of the next best alternative.
- Agents bid with specific attibutes, but commit to providing a certain overall utility. This allows agents to choose alternative means of providing a result, so that the process is more flexible and robust at runtime.
- The possible hazards of allocating utility and allowing result flexibility are mitigated later, during execution, by a failure handling protocol.

The MADSUM negotiation process is designed to reflect the dynamic, user-oriented environment into which it is deployed. Even if a negotiation protocol could be designed that would promise full resource specification and an optimal combination of results, the calculation-intensive solution could be rendered useless by a sudden change in the environment, or a agent's failure to deliver an expected result.

4. UTILITY BASED RESPONSES

First, the top-level Presentation Agent receives a request from the user to provide information that will help in a decision about a proposed investment;[1] this triggers the negotiation process mentioned previously. To estimate the potential significance (Decision Specificity or DS) of the information that they might provide, the lowest level agents utilize a cache of information (saved from prior transactions or gathered during off-peak times) about the proposed investment as well as the user model. The agents will estimate the value they can deliver for each term of the utility function and submit it to their parent agent.

Most agents will submit multiple bids representing a range of information with a range of different resource consumptions and benefits provided. High utility bids are submitted, but also alternative bids that vary widely from the high utility bid in parameter space (this provides options during later parts of the task). This diversity provides flexibility in a dynamic environment. For example, if a user's utility function changes during the message generation, bids that previously had high utility may now be unnattractive. If all the alternatives were chosen based on utility, then *all* might well be unattractive. Bid diversity will also likely be valuable under certain failure circumstances.

At each level of the hierarchy, agents will consider various combinations of the bids submitted in terms of utility, determine a "basket" of combinations, and propagate new bids representing the basket up the tree. This occurs recursively to the top of the tree.

[1]The implemented system does not do natural language understanding; instead the system is given the proposed investment object (such as IBM) and the amount of the investment (such as 100 shares).

2a: Risk metrics indicate IBM has a low debt-equity ratio, suggesting the ability to weather an economic downturn; further, the company has a strong current ratio, indicating good short-term liquidity. In addition, IBM has historically maintained a moderate debt policy, and the stock has maintained a moderate risk profile. On the other hand, from a portfolio perspective you have already exceeded your allocation goal for equities. Value metrics indicate IBM has a price earnings ratio similar to the tech industry average.

2b: Risk metrics indicate IBM has a low debt-equity ratio, suggesting the ability to weather an economic downturn; further, the company has a strong current ratio, indicating good short-term liquidity. On the other hand, from a portfolio perspective you have already exceeded your allocation goal for equities.

2c: Value metrics indicate the stock has a price earnings ratio similar to the tech industry average; on the other hand, from a portfolio perspective you have already exceeded your allocation goal for equities.

Figure 2: Three responses, derived from different soft constraints and priority settings.

Though agents submit multiple bids, each combination includes at most one contribution from each child agent; thus the complexity of the problem is constrained by limiting the number of children of an agent. More precisely, since each basket of bids considered has at most one bid from any given child, then for k children and a maximum number of m bids per child, the number of combinations considered by a task agent is

$$(m+1)^k - 1 \tag{2}$$

Through the hierarchical design of MADSUM the combinatorial auction complexity can be restricted via the number of children a given agent is allowed. There is also work towards making the general case of such auctions more tractable by imposing some minor restrictions on the process; see [18, 3]. Because the auction process is expensive, agents keep copies of bids that are not used, in case of future need (see Section 4.1).

The topmost agent (the Presentation Agent) selects the highest utility bid, which recursively propagates back down the tree as a commitment of resources (see 3.3) and an associated expectation of utility that will be produced.

When the commitment reaches the information agents, they match it to the bid they had made and produce the intended information (consuming resources at the same time). This is the stage that includes transfer of funds to outside agents to cover the cost of any purchased information. The raw information is passed from the lowest-level information agents to their parent task agents, who use the information to generate small text plan trees. As the trees are propagated further up the agent hierarchy, the task agents assemble them using coherence rules for combining text plan trees. In doing so, the task agents first order the text plan trees according to the utility of their highest utility proposition, and the rules for combining trees attempt to assemble larger trees with the higher ranked constituents on the left, so that the higher ranked constituents will appear earlier in the response (subject to coherence constraints). Once an assembled tree is returned to the Presentation Agent, it is resolved to text via templates, and the text is presented to the user.

It is important to note that our system does not seek to provide an "optimal" solution that could be determined by simultaneously considering all possible assemblies of all possible subtrees. (Such an approach would not scale well.) Instead, the system finds the best solution that results from a series of utility-guided choices. Decisions made by agents at every level constrain the decision space of agents above, in theory possibly eliminating the best solutions, but also rendering the communication and calculations practical in size and time.

4.1 Managing failure

We define a "failure" as a result that does not meet expectations. This does not imply that a failed result cannot be used. In many domains a failed result may be better than no result at all; also, a result that is slightly below par when considered by itself may be acceptable in a larger context.

Failure is part of a dynamic world. Information that was valuable during a proposal at time t may not be worth much at $t+1$. Network problems, human errors, and disk failures all cause problems that a distributed system must handle efficiently and with minimal performance degradation.

Failure is a part of the normal operation of MADSUM. The resource allocation issue described above necessitates that "failure" will occur in MADSUM any time agents consider a plan in which two branches want to use the same resources.

The decision to embrace "failure" as a means of addressing the resource allocation problem means that the MADSUM design cannot avoid failure, but must instead focus on being fault-tolerant and minimizing the performance penalties incurred.

MADSUM has four core strategies for managing failures. First, *allow task agents to handle failures that occur below them whenever possible.* This prevents low-level failures from overwhelming the top level agent with failure-related computation.

Second, *agents maintain records of all bids submitted by their children.* When confronted with a failed result from a child, a parent can consider the alternate bids from that child, as well as bids from other children that may not have looked as attractive as the failed bid. This allows an agent to ask a child to fulfill an earlier offer without resorting to a communication-intensive bidding process.

Third, *don't manage failures at the level where they first occur.* Handling a failure may become unimportant when viewed by an agent with greater perspective, either because the result at the next higher level is not substantially adversely affected, or because the failure gains significance in light of other failures and should be propagated still higher. Also, this strategy reduces the possibility that an agent will replace a failed result with one that consumes an inordinate share of resources, since the agent above will be viewing the failed result in the context of other expenditures.

And fourth, *failures that indicate substantial problems with a sub-agent's performance, such as a time-out or exceeding constraints, mean that the sub-agent's other bids will not be considered for failure recovery purposes.*

The failure management protocol in Figure 3 shows the procedure for handling failures caused by insufficient utility being provided by the results from a child agent, as determined by applying the utility function to the attributes of a result. The flow chart shows that low cost attempts to rectify the situation are made early, while solutions that require additional communication or propagation to a higher agent are last resorts. Low cost attempts are based on attribute values, and thus do not require extensive calculation or extensive domain knowledge.

Of primary concern to an agent A is having its result R_A meet the utility expected by its parent P, represented in the utility envelope

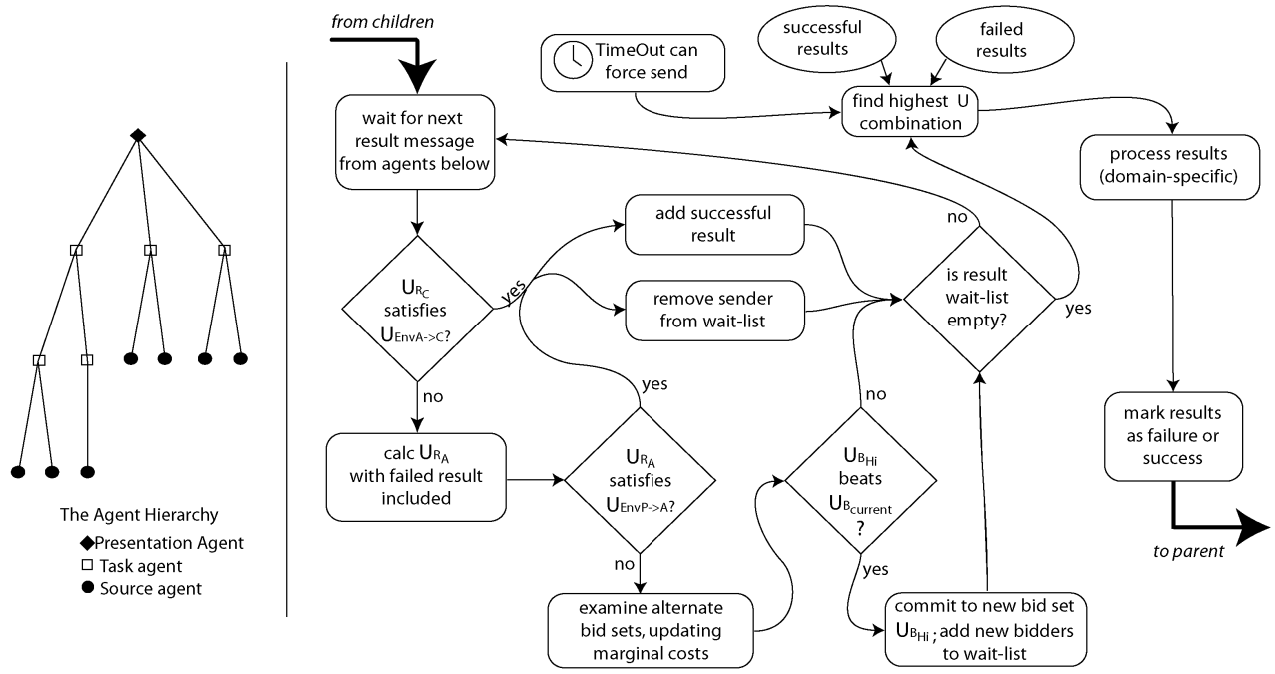

Figure 3: MADSUM current agent hierarchy and protocol for handling failed results.

$U_{EnvP\rightarrow A}$. The result from the child C, R_C, has utility U_{R_C} and is compared to $U_{EnvA\rightarrow C}$ to see if it has failed. If it has, then the protocol proceeds as follows.

The first step of the protocol checks to see if U_{R_C}, though lower than expected, still allows all child results $\sum U_{R_{C_i}}$ to exceed $U_{EnvP\rightarrow A}$. If this is the case, then no further failure processing is necessary unless some other child result fails. The failed R_C may or may not become a part of this agent's result R_A, depending on whether it increases total utility for the result.

Next agent A re-examines every set of bids B_i derived by combining bids from children. For each B_i it is determined whether the set contains a bid already in process, i.e. one to which A has already committed. If A has already committed to a bid, then the marginal resource cost of that bid is zero, thus reducing the apparent cost of B_i. After performing this check on each B_i, A re-calculates utility for each B_i. Then the highest utility (of set B_{hi}) is compared to the (now lowered) expected utility of the bid set $B_{current}$ containing the bid for R_C. If $U_{B_{hi}}$ is *lower* than $U_{B_{current}}$, then the best strategy is to continue the present course. Otherwise if $U_{B_{hi}}$ is greater, then A will commit to the parts of B_{hi} to which is has not already done so.

Note that this marginal cost accounting greatly favors alternatives that include parts that have already been "paid for". Furthermore, because each committment reduces available resources, thus altering constraints, it is unlikely that the system can take a radically different approach after failure unless resources are plentiful.

Finally, when all results are in or a time limit is approaching, the highest utility combination of results is sent for any domain-specific processing that needs to occur. Then results are marked as meeting $U_{EnvP\rightarrow A}$ or failing.

The domain processing happens last since it is expensive relative to the attribute-based utility calculation. Attribute calculations are accurate in many cases (SUM works well for a weight attribute) but may only be a heuristic in others. For example, either SUM or MAX might be correct for combining two height attributes, but it is also possible that a complex 3-D rendering would be necessary. Delaying this calculation until a set of sub-results has been determined heuristically could possibly miss an optimal solution to a task, but also keeps computation costs low.

5. RESULTS

Figure 2 shows three responses produced by MADSUM under different soft constraint and priority settings. The first text was for a user with a soft length constraint of 75 words and a high priority on information about Risk. For the second text, the length constraint was reduced to 35 words while other attributes remained constant. The third text was similar, but with a high priority placed on Value information. Note that the Value information, "similar to the tech industry average" was placed last in the first text and eliminated from the second due to its low DS evaluation. In contrast, information about exceeding a preset goal had a high DS and was included in all three texts even though Portfolio Allocation information was not a high priority.

MADSUM makes decisions about information selection and ordering based on the contents of the user model. It is possible that a user's stated priorities (in the utility function) will not be consistent with the system's analysis of the information's DS based on other aspects of the user model, such as the user's portfolio balances or proximity to retirement. We were particularly interested in 1) the effectiveness of MADSUM's use of a utility function to arbitrate among available propositions for inclusion in a response and 2) the appropriateness of MADSUM's ordering of the selected

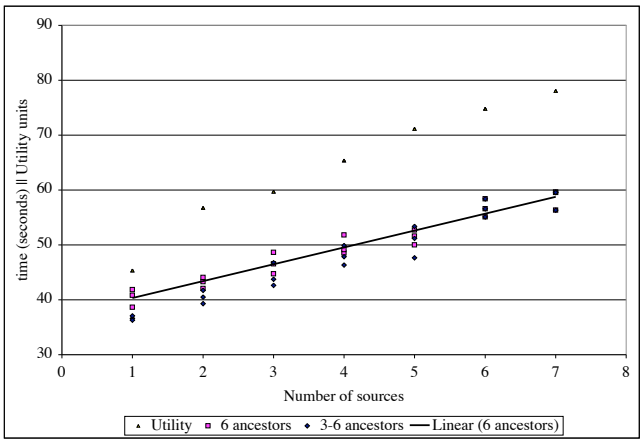

Figure 4: Runtime vs. sources count, 42 trials

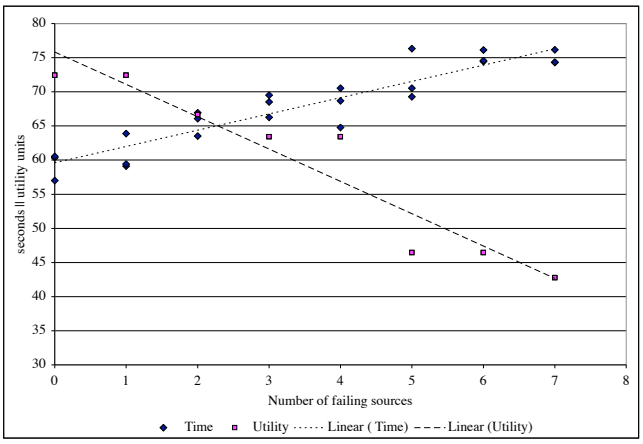

Figure 5: Source failure penalties, 24 trials

propositions.

To test the system's ability to select appropriate content, we performed an experiment in which 21 subjects were each presented with 7 scenarios. Each scenario consisted of 1) a graphic depiction of sliders representing user priorities for the three kinds of content (risk, value, and impact on portfolio allocation goals), 2) two sets of propositions, one of which had been produced by the system. Sometimes the system's response was listed first and other times the alternative appeared first. The subjects were asked to determine, given the user's slider settings for content priority, which content set was most appropriate for that user. In some scenarios, the significance (DS value) of propositions and the priority that the user placed on that kind of information were congruent, (i.e. each proposition was either both significant and high priority, or both of lesser significance and lesser priority) and in other scenarios the significance and priority of propositions were in conflict. The alternatives to the system's responses were constructed to give subjects the opportunity to choose between responses that favored priority in content selection, responses that favored significance, and responses that balanced priority and significance as is done by MADSUM's utility function. We applied a one-tailed binomial test to the results which showed that the subjects had a statistically significant ($p<.01$) preference for MADSUM's strategy of balancing significance and priority in content selection.

MADSUM places the highest utility propositions early in the response, subject to coherence constraints on the construction of text plan trees. To test the system's ordering of propositions in its presentation to the user, we performed an experiment in which 16 subjects were each presented with 9 scenarios. Each scenario again included a graphic depiction of sliders representing user priorities for the three kinds of content. But in this experiment, the subjects were presented with two different orders of presentation of the same propositions. In each case, most cue phrases and connectives were removed in an attempt to prevent subjects from being influenced by phrasing. Once again, a one-tailed binomial test showed statisticallysignificant ($p < .01$) support for the system's decisions about order of presentation of propositions. This was true even when proposition significance (DS value) and user priority for that kind of information conflicted, although the level of statistical significance for this subset of the scenarios dropped to ($p < .05$).

Since multiple MADSUM agents are required to perform a decision support task, it was our intent that the design should scale well. Figure 4 shows that incremental additions of source agents to the system result in a linear increase in time required to perform a task[2]. Data are shown for when only sufficient task agents (three to six) are active above the sources, and also when all six task agents are active throughout the test. Perhaps more importantly, Figure 5 shows the results of the failure management protocol when presented with zero to seven source agents failing to meet utility. Two of the agents had results with utility substantially lower than originally predicted, three had options with similar utility, and two had no result available at all. These linear results are preliminary, and must be repeated with substantially more trials and alternative configurations, but our knowledge of the system leads us to conclude that these performance indications will be borne out.

6. RELATED WORK

MADSUM is related to work in decision support applications, multi-agent systems, user modeling, and failure recovery. [12] is a single-agent information gatherer that provides decision support and responds to user priorities, but the priorities are related to the problem domain, while MADSUM, a distributed system, can also consider preferences about the attributes of the response itself, such as length.

Other systems have made use of utility theory to take user preferences into account. Early work using multiple attribute preferences with adjustable factors includes [2]. Lesh used a similar model in ranking airline flights for a travel domain. More recent work includes MATCH[17] and FLIGHTS [14] which use formal utility functions to allow user preferences about domain attributes to influence system results.

While each of these systems makes use of a utility function, they each use the utility function to compute the utility of the expected *result* of a choice by the user - the utility of a certain train ride, or a particular airline flight. In contrast, MADSUM's utility function

[2]For timing experiments all agents ran on a single laptop.

explicitly includes user influence on attributes of the system's output, in terms of cost, length, and time, as well as domain-specific preferences such as topic. Thus MADSUM agents are concerned with the total utility of the *message* to the user, including the utility of the both information provided as well as the the utility of the attributes of the presentation. In addition to placing the focus of the system on the value of the message to the user, this allows the system to represent a high-utility message about a low-utility choice.

Our decentralized failure management protocol is largely domain-independent and relies on a team of cooperative agents, as do [11] and [9]. However, these systems focus primarily on agent failure, and assume a high degree of inter-agent connectivity. MADSUM failure management emphasizes problems with the quality of results, and all communication is along the tree structure.

7. CONCLUSIONS

MADSUM is an implemented multi-agent system for decision support. It was designed to provide integrated texts that reflect both the information and resource priorities of a user while functioning in a dynamic information environment. A human study shows significantly ($p < .01$) that human subjects agreed with the way MADSUM balances user priorities and information significance when selecting information for presentation. A second study shows agreement ($p < .05$) with MADSUM's method of ordering information even when information importance and user priority are not congruent. While these tests cannot fully predict the ultimate quality and usefulness of MADSUM ouput, information selection and ordering are two critical tasks in presenting information for decision support.

MADSUM is an adaptive system that relies on a negotiation process to acquire and integrate information from multiple sources. The architecture was designed for an environment where information utility could not be easily predicted *a priori*. Thus a small number bids from each agent allow the establishment of a target utility at a high level, while MADSUM agents allocate expected utility recursively to sub-agents, allowing the sub-agents to determine precisely how to derive that utility themselves. This allows for several options in handling information failures, each with a different associated cost. Preliminary testing shows that MADSUM's efficient failure management protocol handles the number of failures in linear time.

8. REFERENCES

[1] D. N. Chin and A. Porage. Acquiring user preferences for product customization. In *UM '01: Proceedings of the 8th International Conference on User Modeling 2001*, pages 95–104. Springer-Verlag, 2001.

[2] J. Chu-Carroll and S. Carberry. A plan-based model for response generation in collaborative task-oriented dialogues. In *Proceedings of the 12th Annual American Association for Artificial Intelligence*, pages 799–805, 1994.

[3] W. Conen and T. Sandholm. Anonymous pricing of efficient allocations in combinatorial economies. In *AAMAS*, pages 254–260, 2004.

[4] K. S. Decker and V. R. Lesser. Quantitative modeling of complex computational task environments. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 217–224, Washington, July 1993.

[5] K. S. Decker and K. Sycara. Intelligent adaptive information agents. *Intelligent Information Systems*, 9:239–260, 1997.

[6] Foundation for Intelligent Physical Agents (FIPA). Fipa '97 specification part 2: Agent communication language. `http://drogo.cselt.stet.it/fipa/`.

[7] J. Graham, K. Decker, and M. Mersic. DECAF - a flexible multi-agent system architecture. *Autonomous Agents and Multi-Agent Systems*, 7(1–2):7–27, 2003. Overview of the DECAF architecture.

[8] N. R. Jennings. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artificial Intelligence*, 75(2):195–240, 1995.

[9] G. A. Kaminka and M. Tambe. What is wrong with us? improving robustness through social diagnosis. In *AAAI '98/IAAI '98: Proceedings of the fifteenth national conference on Artificial intelligence*, pages 97–104. American Association for Artificial Intelligence, 1998.

[10] F. Klusch, Matthias; Zambonelli, editor. *5th International Workshop on Cooperative Information Agents*, Lecture Notes in Computer Science 2182, Modena, Italy, September 2001. Springer.

[11] S. Kumar and P. R. Cohen. Towards a fault-tolerant multi-agent system architecture. In *AGENTS '00: Proceedings of the fourth international conference on Autonomous agents*, pages 459–466. ACM Press, 2000.

[12] V. Lesser, B. Horling, F. Klassner, A. Raja, T. Wagner, and S. X. Zhang. Big: an agent for resource-bounded information gathering and decision making. *Artif. Intell.*, 118(1-2):197–244, 2000.

[13] G. Linden, S. Hanks, and N. Lesh. Interactive assessment of user preference models: The Automated Travel Assistant. In A. Jameson, C. Paris, and C. Tasso, editors, *User Modeling: Proceedings of the Sixth International Conference, UM97*, pages 67–78. Springer Wien New York, Vienna, New York, 1997. Available from http://um.org.

[14] J. Moore, M. Foster, O. Lemon, and M. White. Generating tailored, comparative descriptions in spoken dialogue. In Proceedings of FLAIRS-04, 2004.

[15] C. J. Petrie. Agent-based engineering, the web, and intelligence. *IEEE Expert*, December 1996.

[16] K. Sycara, K. S. Decker, A. Pannu, M. Williamson, and D. Zeng. Distributed intelligent agents. *IEEE Expert*, 11(6):36–46, Dec. 1996.

[17] M. Walker, S. Whittaker, A. Stent, P. Maloor, J. Moore, M. Johnston, and G. Vasireddy. User tailored generation in the match multimodal dialogue system. In *Cognitive Science*, volume 28, pages 811–840, 2004.

[18] P. R. Wurman and M. P. Wellman. A$k$ba: A progressive, anonymous-price combinatorial auction. In *Second ACM Conference on Electronic Commerce*, pages 21–29, Minneapolis MN, October 2000.