DECISION SUPPORT COMMUNICATION: INTEGRATING COMMUNICATIVE PLANS FROM MULTIPLE SOURCES TO PLAN MESSAGES FOR A DYNAMIC USER AND ENVIRONMENT

by

Terrence Harvey

A dissertation submitted to the Faculty of the University of Delaware in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science

Fall 2006

DECISION SUPPORT COMMUNICATION: INTEGRATING COMMUNICATIVE PLANS FROM MULTIPLE SOURCES TO PLAN MESSAGES FOR A DYNAMIC USER AND ENVIRONMENT

by

Terrence Harvey

Approved: _____

Sandra Carberry, Ph.D.
Professor of Computer Science

Approved: _____

Keith Decker, Ph.D.
Professor of Computer Science

Approved: _____

Daniel Rich, Ph.D.
Provost

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Sandra Carberry, Ph.D.
Professor in charge of dissertation

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Keith Decker, Ph.D.
Professor in charge of dissertation

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Peter Cole, Ph.D.
Member of dissertation committee

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Owen Rambow, Ph.D.
Member of dissertation committee

ACKNOWLEDGEMENTS

I had a great deal of help and support from dozens of faculty, fellow students, family, and friends. Thanks to every one of you.

My experience with my committee was fabulous. Thanks to Peter Cole for thinking outside the NLG box; to Owen Rambow for making me more interested in my work every time we met; and to Keith Decker, who showed me agents and can "blue sky" with the best of them. Most of all, thanks to Sandee Carberry, who was there from the very start, who kept me focused and grounded, and who convinced me that I could finish.

Finally, thanks to Line Farr for her amazing patience and support during a long haul.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

When computers communicate with humans to facilitate a decision process, they must do so in an unambiguous and concise manner which reflects the needs and preferences of the user, and takes into account the situation in which the system performs. This dissertation explores the fulfillment of these requirements of decision support in the context of systems that receive information from multiple, independent sources. In the first part of this thesis I present my hypothesis that independent text plans, analyzed in terms of their rhetorical structure[MT87], can be integrated to enhance conciseness and coherence. An implemented system, *RTPI*, demonstrates the application of this hypothesis to improve message sets generated by an existing decision support system through the integration of related messages. An evaluation of *RTPI* shows that it reduces repetition and overall message length, and human subjects prefer its integrated messages to the original sets of messages.

The second part of this thesis explores the additional challenges presented when decision support information is obtained and integrated from multiple sources in a dynamic information environment. In this environment, information attributes such as quality, cost, length, and production time can vary over time; in addition, user preferences regarding those attributes can also change. I present my hypothesis that this problem can be computationally addressed, by a flexible, responsive, agent-based system that takes into account both user preferences and information attributes that change over time. The implemented system *MADSUM* demonstrates a multi-agent architecture that addresses all of these issues. An evaluation of the system's output shows that human subjects,

when presented with a scenario including user preferences and information, agree with *MADSUM*'s selection and ordering of information from a proposition set.

Chapter 1

COMMUNICATION FOR DECISION SUPPORT

1.1 Introduction

A primary strength of computers is their ability to perform calculations on large amounts of data quickly. Not long after commercial computers were developed, people started using them to process information in ways designed to help people make decisions when analysis of data could theoretically improve the decision process[Mor67, Pow03]. Involving computers in a decision process requires humans to communicate with computers, and vice versa.

Decision support has special requirements of communication. To be useful to a particular human, communication must take into account user preferences. A user may have preferences regarding performance parameters, for example that the answer be available in a reasonable period of time at some reasonable cost. But communication specific parameters may also be important, such as the number of pages of output, use of domain-specific jargon, etc. Furthermore, individual users may have distinct priorities regarding the content of communications.

Communication for decision support must also be provided in a useful and easily assimilated form. Grice's maxim of manner[Gri75] suggests that communication should be unambiguous and concise, so messages should not be conflicting or repetitive. A decision support system that generates files of numbers may not be incorporated into a human decision process as readily as one that provides natural language summaries. Within systems that produce natural language, a system that uses a style that is appropriate for its audience is preferable to a system that uses a single style for all communications.

In order to be useful, and therefore incorporated into human decision processes, decision support communication must both adapt to the needs and preferences of the user and produce results that are appropriate for the situation in which the system performs.

This thesis is the result of two related investigations into communication for decision support. Both explore the problems found when decision support information provided to a system comes from multiple, independent sources. The first part of my thesis results from a need to improve the quality of messages designed for use by a physician in a trauma setting. Sets of messages produced by different modules of an existing decision support system, TraumaTiq, displayed messages that sometimes were repetitive and appeared to conflict with one another. I hypothesize that analysis of the messages in terms of Rhetorical Structure Theory[MT87] will allow the message sets to be computationally transformed to enhance conciseness and coherence.

The second part of my research investigates the challenges presented when decision support information is obtained and integrated from multiple sources in a dynamic information environment. In such an environment, information attributes such as quality, cost, length, and production time vary over time; furthermore, user preferences regarding those attributes can also change. I hypothesize that this problem can also be computationally addressed, by a flexible, responsive, agent-based system that takes into account changing user preferences on dynamic information attributes.

I have developed two systems to explore the nature of decision support communication and to test these hypotheses. The first, *RTPI*, is designed to address problems in the messages produced by the TraumaTiq system. The second system, *MADSUM*, is designed to produce decision support messages in a dynamic environment.

1.2 *RTPI*: Rhetorical Text Plan Integrator

The messages produced by TraumaTiq are intended to be read by a physician operating with a medical team in a trauma bay. However, the messages, while individually correct, exhibit various problems when presented in sets. For example, the two shown in

Two messages produced by TraumaTiq:

⋆ Caution: check for medication allergies and order pulmonary care immediately to treat the left pulmonary parenchymal injury.

⋆ Caution: check for medication allergies and order pulmonary care immediately to treat the compound rib fracture of the left chest.

Figure 1.1: Simultaneous communications from TraumaTiq showing repetition of "check for medication allergies", and "order pulmonary care".

Figure 1.1 repeat two actions. Repetition could slow the assimilation of the messages by the physician, or may even suggest multiple performances of an action when only a single action was intended. Other messages generated by TraumaTiq appeared to conflict with each other.

I hypothesize that re-organizing the messages could make the message sets more concise and coherent. More specifically, I hypothesize that analyzing the messages in terms of their rhetorical structure[MT83] will provide a framework in which the sets can be transformed to enhance them, while preserving their original communicative intent and respecting the system's social role in the trauma bay; and that these transformations can be performed computationally.

1.3 *MADSUM*: Multi-Agent Decision Support with User Modeling

Zack [Zac04] investigated four attributes of information in a decision process: uncertainty, complexity, ambiguity, and equivocality[1]. Comparing human and computer-based methods of making decisions, Zack found that while human-based strategies (such as meetings) are better at addressing problems featuring ambiguity and equivocality, computer decision support appears more appropriate when situations involve uncertainty and complexity. It is exactly these characteristics that are present in the kinds of domains

[1] "Equivocality" is present when choices are very similar to one another.

for which *MADSUM* is intended. Uncertainty is present since information and information attributes change over time, rendering communication plans susceptible to failure. Complexity is present because of the variety of kinds and sources of information that are presented to the user who is making the decision. Thus computer decision support is appropriate in these domains.

My hypotheses are that a computational methodology can be developed to produce and integrate communication plans for decision support under changing conditions, specifically

- user preferences that change over time;
- an environment in which it is difficult to predict the resources consumed by an information gathering action; and
- an environment in which it is difficult to predict the attributes of information gathering results.

Furthermore, such a methodology should perform quickly in the presence of such changing conditions so that computational decision support in a dynamic environment is shown to be a realistic goal.

Chapter 2 examines TraumAid, TraumaTiq, and their messages more completely. It elaborates upon the hypotheses explained in Section 1.1, and then introduces *RTPI* and explains how I designed the system to implement my hypotheses. Chapter 3 presents realization features and concludes with the results of an evaluation of the original TraumaTiq messages with the transformed messages as produced by *RTPI*.

Chapter 4 introduces *MADSUM*, which I designed to address the hypotheses presented in Section 1.3. Chapter 5 elaborates on *MADSUM*'s decision-theoretic approach to response generation that produces responses tailored to an individual user. It concludes by presenting the results of evaluations of *MADSUM*'s content selection and the ordering of propositions in *MADSUM*'s messages, both of which reflect the preferences of the

user. In Chapter 6, I explain how the architecture of *MADSUM* is designed to address the issues presented by changeable user preferences and a dynamic information environment. The chapter concludes with a series of evaluations that investigate the architecture's performance and response to changes in the environment, including changes that cause communicative plans to fail.

Chapter 7 briefly summarizes the previous chapters on *RTPI* and *MADSUM*. I present both my hypotheses and the conclusions I have drawn from my work. Finally, I discuss possible future directions for my research.

Chapter 2

THE RULE-BASED TEXT PLAN INTEGRATOR

2.1 Introduction

Computers are useful because of their ability to keep track of huge amounts of information and make quick calculations. To the extent that we can relate decision-making to calculations, we make computers useful in human decision support. This usefulness is limited by our (in)ability to design computer programs that can be integrated into human systems. One human system that could benefit from a computer-based decision support system is the medical team working in a trauma center. A trauma surgeon has to make decisions about a patient whose condition may be changing rapidly, while keeping track of voluminous patient data and the highly complex, inter-related guidelines for choosing medical procedures.

The TraumAid system was designed to fill this need. TraumAid [WCC+98] is a decision support system for addressing the initial definitive management of multiple trauma. Initial evaluation and validation studies indicate that TraumAid produces high-quality diagnostic and therapeutic plans for managing patient care in both simple and complex trauma cases [GWC+97]. While the quality of TraumAid's planning is excellent, to be integrated into a human trauma team the system had to be able to communicate with the team. Communicating the entire plan was not likely to succeed in a noisy time-critical environment, so it was decided that only the differences between the computer generated plan and the inferred physician plan would be communicated [WCC+98].

TraumaTiq [GW96] is a module that is designed to provide the physician with real-time messages about how the physician's plan compares to TraumAid's plan. Though the

A set of critiques produced by TraumaTiq:

⋆ Caution: check for medication allergies and order pulmonary care immediately to treat the left pulmonary parenchymal injury.

⋆ Caution: check for medication allergies and order pulmonary care immediately to treat the compound rib fracture of the left chest.

⋆ Caution: check for medication allergies and do a laparotomy immediately to treat the intra-abdominal injury.

⋆ Caution: do a laparotomy and repair the left diaphragm immediately to treat the lacerated left diaphragm.

⋆ Consider checking for medication allergies now to treat a possible GI tract injury.

Figure 2.1: Multiple, simultaneous critiques from TraumaTiq showing repetition of three procedures: checking medication allergies, ordering pulmonary care, and doing a laparotomy.

---

messages from TraumaTiq are individually concise and coherent, they are almost never presented singly. Instead, TraumaTiq presents a *set* of critiques, and when multiple critiques share subject matter they sometimes appear redundant, disorganized, or incoherent, which may inhibit rapid message assimilation by a trauma physician [CH97].

For example, Figure 2.1 shows five messages that contain three instructions to "check for medication allergies", two instructions to "order pulmonary care" and two to "do a laparotomy". Each instruction is separately motivated; however, the system is *not* suggesting that the physician check for allergies three times, or perform two laparotomies. Therefore such repetition should be avoided if possible to conserve time if reading critiques, conserve space if presenting written critiques, avoid taxing the physician's attention unnecessarily in either case, and make the recommendations of the decision support system easier to assimilate as a whole. [1]

My research hypothesis is that an analysis of the texts produced by TraumaTiq in terms of their rhetorical structure [MT83] will provide a framework in which sets of

[1] Unless perhaps a system *chooses* to deliberately use repetition as a rhetorical means of emphasis, which is not the case here.

these texts can be re-organized so that they are collectively more concise and more coherent. Working within this framework will ensure that the original intent of the TraumaTiq messages is preserved [MT87]. More specifically, my hypotheses are that:

- the collective texts can be made more concise by reducing repetition;
- messages that currently appear incoherent can be re-organized so that they are more coherent; and
- these tasks can be performed computationally by designing a set of rules to perform the re-organization on the critiques produced by TraumaTiq.

To phrase it more technically, if each of the original messages from TraumaTiq is viewed as a communicative plan with a goal of communicating something to the physician, then I hypothesize that it is feasible to develop a system to integrate a set of those plans. The resulting integrated text plan must retain and express each of the individual plan goals and the means to achieve those goals, while increasing coherence and conciseness relative to the original separate TraumaTiq messages and respecting the social role of the system in the trauma bay. It is my intent and assumption that this will make the messages more effective in a trauma setting, though this has not been empirically verified.

My approach to this problem is presented in the implemented system *RTPI* (Rule-based Text Plan Integrator). I developed *RTPI* to address problems in the output of the medical decision support system, TraumaTiq. RTPI draws on rhetorical structure and discourse theory to integrate individual message text plans, each of which is designed to achieve a separate communicative goal. The system takes as input a *set* of individual text plans represented as RST-style trees, and produces a smaller set of more complex trees representing integrated messages. Domain-independent rules are used to capture text plan integration strategies, and the rule set is extendable. The system has been tested on a corpus of critiques in the domain of trauma care.

The remainder of this chapter describes the theoretical aspects of the approach taken in the development of *RTPI*, a system that works with the messages produced by the decision support system, TraumaTiq, and implements my proposed solutions to the problems described above. Section 2.2 presents the motivation and previous work behind *RTPI*, and Section 2.3 contrasts it with other work. Section 2.4 presents an overview of the system, and Section 2.5 introduces the task of integrating multiple communicative goals. The process by which *RTPI* performs these integrations is described in Section 2.6, which also introduces *RTPI*'s domain-independent rules for integrating text plans. Section 2.7 then summarizes key points of the chapter.

2.2 TraumaTiq: Decision Support in Real Time

TraumaTiq [GW96] informs the physician about discrepancies between the physician's plan and TraumAid's plan. TraumaTiq achieves this by hypothesizing the physician's plan based on his or her orders and actions, which are already being recorded by the hospital for other purposes. An inferred physician plan for managing patient care is then compared with TraumAid's own management plan. Discrepancies between the plans are identified by TraumaTiq, which then generates a set of messages. In particular, TraumaTiq recognizes four classes of differences:

1. errors of omission, where the physician has not ordered an action that TraumAid would have ordered;

2. errors of commission, where the physician has ordered actions that TraumAid does not have in its plan;

3. scheduling errors, which are actions ordered in the absence of orders for preconditions to that action; and

4. procedure choice errors, when the physician orders an action that TraumaTiq recognizes as addressing an appropriate goal, but the procedure chosen is suboptimal.

Two TraumaTiq messages:
Performing local visual exploration of all abdominal wounds is preferred over doing a peritoneal lavage for ruling out a suspicious abdominal wall injury.

Please remember to check for laparotomy scars before you do a peritoneal lavage.

Figure 2.2: An example of two messages with apparent conflict about the performance of a peritoneal lavage.

To hypothesize the physician's plan, TraumaTiq first chains through TraumAid's knowledge/rule base to identify possible explanations for an action ordered by the physician; it then evaluates these possible explanations on the basis of relevance to TraumAid's current management plan and evidence provided by the physician's other actions (ordered or performed). Once the best explanation(s) for each action have been incorporated into the system's model of the physician's plan, TraumaTiq identifies differences between that inferred plan of action and TraumAid's current management plan and notifies the physician of discrepancies that could seriously impact patient care. By inferring the physician's plan, TraumaTiq can take into account *why* the physician is performing a particular action when deciding 1) whether or not to produce a message informing the physician of the discrepancy between TraumAid's plan and the inferred physician plan, and 2) if so, what information to include in the message. TraumaTiq's messages are conveyed using natural language sentences generated by filling in sentence schemata.

In isolation each of TraumaTiq's messages may effectively warn a physician about a problem in the inferred physician plan; however in most cases when TraumaTiq finds the physician's plan deficient, several problems are detected simultaneously and thus multiple messages are produced. In these cases there were three substantial kinds of unintended interaction between the individual messages:

Original TraumaTiq messages:
Caution: do a peritoneal lavage immediately as part of ruling out abdominal bleeding.

Do not reassess the patient in 6 to 24 hours until after doing a peritoneal lavage. The outcome of the latter may affect the need to do the former.

Figure 2.3: These two inter-dependent critiques from TraumaTiq do not conflict with one another, but could be expressed more clearly if their relationship to one another was exploited.

---

1. Some messages detracted from other ones, or could appear incoherent in the presence of other messages. For example, one message set might contain a message "Don't do A" and another message "Do B before you do A." as in Figure 2.2. The implication of the second message is that TraumaTiq approves of the performance of A, while the first clearly indicates the action should not be performed. This apparent conflict is the result of the individual messages being produced by separate sources within TraumaTiq.

2. There was informational overlap among the messages. For example, the individual messages in Figure 2.1 share references to physician actions. Each reference was made for a different original purpose (e.g. to note an action missing from the plan for addressing a particular treatment goal, or an action whose preconditions had not been met), but repetitions of an action's name across individual messages unnecessarily increases the overall message length, and may slow the physician's assimilation of the message. Furthermore, multiple references to a single action may be erroneously interpreted as suggesting that the action be performed multiple times.

3. Some messages would make more sense if they took explicit account of those appearing earlier. The example in Figure 2.3 shows two messages that express information about a peritoneal lavage. One message says that the lavage should be performed and why, and the other message expresses that a scheduling constraint depends on the result of the lavage. If these messages could be integrated, the relationship between them could be exploited to form a clearer message.

In addition, sometimes messages appeared to be about the same topic, but were widely separated when the TraumaTiq message set was presented. These will be further discussed in Section 3.2. Thus a message planner was needed to generate coherent and concise integrated messages that satisfy the multiple goals of the individual messages.

A corpus of actual cases of trauma care was used as input to TraumaTiq, and the resulting sets of messages became a corpus for *RTPI*. A total of 753 sets were produced, consisting of a total of 5361 individual messages. Analysis of half of this data (the remaining half was reserved for later evaluation of performance on unseen data) and the semantics underlying the message forms revealed 22 common patterns of inter-related messages, each pattern covering some subset of a message set.

However, it became clear that many of the patterns are more generally applicable, and that the problems addressed would not be unique to TraumaTiq, or even a medical domain. The same patterns of message interaction could also arise when other distributed systems had multiple, separate modules that needed to communicate inter-related messages to a single user. While such systems could be re-designed to try to prevent problems of this kind from arising, the result would be less modular, more complex, and more difficult to extend and maintain. Thus *RTPI* was developed as a message integrator that takes message sets as input and uses patterns encoded in domain-independent rules to improve them. The message sets from TraumaTiq are first expanded to full representations of RST-style plan trees, so that *RTPI* is not dependent on the specific logical representation of messages made by TraumaTiq.

2.3 Related Work

Although there is related work in the areas of text structure, text planning for generation, text plan revision, instruction generation and multi-document summarization, little attention has been given to assembling or integrating multiple, independently generated text plans when each is intended to function as one component of a larger message.

Fundamental to my development of *RTPI* is a theory of the functional analysis of text called Rhetorical Structure Theory, developed by Mann and Thompson [MT83, MT87]. According to RST, a text contains *relational propositions* that are conveyed by the way the text is structured. The structure of a text (derived from the arrangement of clauses and the presence of discourse markers) is necessary to determine meaning, and the meaning of a text cannot be derived from simple composition of the semantics of lexical items.

The relational propositions that exist in the text are represented in the RST formalism by *relations*. According to Mann and Thompson, relational propositions possess the following properties:

- In most cases relational propositions are not explicitly expressed in a clause.
- Relational propositions can be signaled via discourse markers, but often are not signaled at all.
- One relational proposition corresponds to one RST relation in the text structure.
- Relations are capable of performing rhetorical acts, such as changing a reader's beliefs or desire to act.
- Finally, "The relational propositions are essential to the coherence of their texts. Perturbing text to prevent the (implicit or explicit) expression of one of its relational propositions causes the text to become incoherent.[MT87]"

A single RST relation holds between adjacent text spans, and relations are defined by a list of constraints and effects. Most relations are defined in terms of a nucleus text span and a satellite text span, where the nucleus is a span that remains comprehensible in the absence of the satellite. Restrictions on the way that relations are applied ensure that coherent texts (with a few special exceptions) will have an RST structure that is a tree of text spans joined by relations. Two contributions of RST that facilitate text generation are 1) the idea that recognizing the relations in an RST structure is equivalent to finding the basis for the text's coherence, and 2) RST works at both the inter-clause and inter-sentence levels, allowing a single planner to plan both sentences and paragraphs.

Alternatives to RST exist that propose other means for structuring text, including Grosz and Sidner[GS86], Scha and Polanyi [SP88], and Moore and Pollack [MP92]. Moore and Pollack showed that some texts have more than a single relation holding between two spans of text, and argue that two separate structures, possibly non-isomorphic, are required to represent separate informational and intentional relations between text spans. Such texts cannot be accurately represented in RST, which allows only a single relation to hold between spans. While this point is adequately demonstrated for the text cases Moore and Pollack provide, the kinds of texts used as input for *RTPI* have simpler structures that are accurately represented within the restrictions of RST. Similarly, the output of *RTPI*'s rules also fits within the RST paradigm, and so does not justify the overhead of additional structures for representing text plans.

Both Polanyi and Scha's Linguistic Discourse Model (LDM) and the structures of Grosz and Sidner's work support the analysis of features of discourse that are not explained by RST. Like RST, the LDM builds a tree of text spans (called *Discourse Constituent Units* or DCUs) where the text appears at the leaves in the order of the original text. A grammar provides rules for adding new sentences to the active right edge of the tree, and the resulting structure facilitates tasks such as anaphora resolution and information interpretation. Grosz and Sidner's work posits three separate components in

discourse structure: a *linguistic* structure that holds the structure of a sequence of utterances; an *intentional* structure that captures the discourse purposes of the segments and relationships among them; and an *attentional state* which tracks the participants' focus of attention. These structures provide a basis for explaining such aspects of discourse as cue phrases, anaphora, and interruptions, and have been applied to conversations as well as single source texts. Thus both the LDM and Grosz and Sidner's work explain phenomena that are not explained by RST. However, neither work provides a set of specific relations that hold between text spans, such as RST's ELABORATION or CONTRAST which are defined in terms of the writer's desired effect upon the reader. These specific relations (as opposed to more general relations like "subordinate") are what enable the construction of *RTPI*'s rules.

Thus while alternatives to RST exist for structuring text plans, they are intended to handle features of text that do not occur in *RTPI*'s domain. However, it is worth noting that if *RTPI* is modified to work in a new domain, then it could be the case that either the text inputs of that domain would require handling of text features that are not represented in RST, or that such features would be required to improve the text inputs. In that case, the RST formalism would not suffice and alternatives would have to be considered.

Hovy [Hov88, Hov91] exploited the properties of RST relations, using them to construct text plans that were coherent because they had been built using the principles of RST. Hovy also developed text planning operators (rules that could build a single text plan) based on RST relations, and used them to plan text structures from the top down. Each operator used constraints to determine what content could be used to fill the nucleus and satellite roles of a given RST relation. Operators also had growth points, optional ways to extend the structure that could be posted as further goals for the planner. The user's communicative goal was achieved by matching it against the RESULTS field of an operator. The operator posts sub-goals, and the process continued so that a tree of relations is built with small text spans as leaf nodes.

Hovy's planner used operator constraints to select content to address a top-level goal, and then used the growth points to add as much additional information as possible. Thus only the first step of the process was directly related to the top-level goal of providing a particular piece of information. Successive steps were influenced by the available data[2], since that determined which growth points worked, and by the choice of growth points associated with an operator. Because the planner selected the text plan that included the greatest number of propositions, it always "said" as much of the data as possible; thus an explicit goal of expressing all knowledge wasn't necessary. On the other hand, it was not possible to ensure that all data on a subject would be expressed.

Moore and Paris [MP90, MP93] also used RST relations to do top-down text planning. Their operators matched their effects against a goal posted above, and posted a nucleus and satellite combination designed to satisfy that goal. The nucleus and satellite became the new goals (until they became atomic actions and were designated leaf nodes). Like Hovy's growth points, the "knowledge" of how the tree could expand below a given goal was determined by the operator's nucleus/satellite combination. Unlike Hovy's planner, the Moore and Paris planner only employed an operator when it was necessary to satisfy a goal. Thus their system generated only enough text to satisfy the top-level goal, and favored concise trees over larger ones.

Moore and Paris recognized that for the resulting plan trees to be useful in a dialogue system, the plans would have to include not only the relations between spans of text, but also the intended effect that the chosen relation was supposed to have on the hearer. This property facilitates the process of designing a system response when the intended effect does not occur as planned.

ILEX [MOOK98] is a bottom-up, opportunistic RST-based planner. Objects in the knowledge base are linked via RST-style relations, and the system constructs a tree by

[2] The "available data" was a set of clause-sized, structured information units extracted from a less structured database using domain-specific rules.

searching links starting from the desired topic. The structures are limited to relations that are already known to exist between items in the knowledge base. In contrast, my text plan integrator can superimpose certain new relations, such as CONCESSION, between parts of existing trees; this can be done because the new relation doesn't represent domain concepts, but a relation between the intentions in the tree structures.

Marcu [Mar97a, Mar97b] also builds RST-style structures in a bottom up fashion, and in contrast to ILEX, can have a top-level communicative goal of communicating all of the propositions in a knowledge base. Like ILEX, however, the system depends on having all relations between propositions encoded into the knowledge base. If some subset of the knowledge base is not connected to the rest of the knowledge via pre-coded relations, then a goal to communicate all of the information will not succeed. (In contrast, *RTPI* integrates messages between which there is no pre-existing relation, since the messages need only fit the tree structure patterns required by a rule.) Furthermore, while the plan may express every proposition in the knowledge base, it does not attempt to express every relation between propositions in the knowledge base; so information contained in the *relations* of the knowledge base can be lost when a proposition is related to more than one other proposition in the knowledge base. For example, suppose two propositions P_1 and P_2 are related by the RST relations SEQUENCE (meaning they occur in sequence) as well as NV-CAUSE (meaning that one of the propositions caused the other). Then Marcu's planner can be required to present both P_1 and P_2, but cannot be required to present that they are related via both SEQUENCE and NV-CAUSE. In contrast, *RTPI*'s rules express all relations present in the messages from TraumaTiq.

Existing RST-based generation systems, like those above, function in explanatory or descriptive environments, not decision support, and the difference in environment was reflected in the tasks the systems performed. For example, while Moore and Paris' system had plan operators to convince a user to perform an action, decision support can (and *RTPI* does) require operators to convince a user that they should *not* do a certain act.

Another difference between the decision support environment and that of the RST-based explanation systems is that the latter assume that system and user beliefs follow the overlay assumption: the beliefs and knowledge of the system are a superset of those of the user. Decision support cannot use this simplifying assumption, since it must contend with conflicting beliefs and goals. In particular, *RTPI* has to integrate plans that are designed to convince a user to adopt a goal, or to abandon a previously held goal.

Domain-independent text planning rules like the ones used by my text plan integrator are not a new idea. Appelt [App85] used "interactions typical of linguistic actions" to design critics for action subsumption in KAMP. After a plan had been generated from a single top-level goal, the plan would be "reviewed" and the critics could opportunistically subsume small clauses. If there were two communicative goals `(KnowsWhatIs John Tool(B1))` and `(Active Tool(B1))`, both could be satisfied by a single mention of `Tool(B1)`. This worked in part because many of KAMP's goals (e.g. KnowsWhatIs and Active) could be satisfied simply by mentioning the name of an item, so that the two goals mentioned here, which occurred several times each in the plan, were satisfied by the appearance of the word "wrench" in "Remove the pump with the wrench in the toolbox". Thus a single referring act could satisfy multiple communicative goals, though the system was limited to spans of one or two sentences. *RTPI*'s integration rules, instead of working only within clauses, work between text plans that encompass sentences and paragraphs.

HealthDoc's data-driven text planner is also rule-based[DHH97, HDHP97]. It starts with a pre-written, coherent "master document" that communicates a consumer health message, then modifies the document for a particular patient model while attempting to maintain coherence. The process focuses on editing sentences selected from the master text, such as by inserting pronouns or by deleting references to propositions that do not appear earlier in the selected text[WH96]. Its rules can remove redundancy by

aggregating neighboring expressions, but it does not address the aggregation of communicative goals (often requiring reorganization), the revision and integration of text plans to remove conflict, or the exploiting of relations between communicative goals as done by my text plan integrator. In contrast, my integrator's rules examine the communicative goals in a set of text plans, and certain rules may aggregate or subsume goals, or insert new goals in the process of improving the set's coherence and conciseness.

REVISOR [CL97] is based on RST. As the name suggests, it is not a full planner, but makes revisions (in the form of clause aggregation in a descriptive or explanatory text) to an existing text plan. REVISOR's input did not include plans that appeared conflicting because of text structure; instead REVISOR's primary task was to reduce repetition through aggregation of neighboring clauses. An important similarity to my work is that REVISOR makes its modification decisions without detailed semantic or lexical knowledge of low level constituents. However, to achieve real-time performance requirements as part of a life-like animated agent system, REVISOR uses a minimalist representation of a discourse plan that removes all but the most essential semantic features for processing. While determining essential features is possible with any given set of aggregation rules, it is not possible for a system that could incorporate new rules at any time, where new rules could require information that was not essential to the performance of the previous set of rules. In contrast, *RTPI* uses the entire plan tree, allowing the introduction of new rules without the concern that information previously considered unimportant will be missing. And because my rules operate on full RST-style text plans that include communicative goals, the rules can be designed to integrate the text plans in ways that still explicitly satisfy those goals.

WISHFUL [ZM93, ZM95] includes an optimization phase during which it chooses the optimal way to achieve a set of related communicative goals. However, the system can choose to eliminate propositions and does not have to deal with potential conflict within the information to be conveyed. Similarly, STREAK [Rob94] is not required to include

all information in a plan. Its rule-based text plan revision component is given a single text plan containing "obligatory" facts, along with a set of optional "supplementary" facts to be opportunistically included if realization constraints are met. In contrast, *RTPI* operates on multiple text plans *after* a domain expert has determined what information is vital to the communication.

Work in developing instructional texts [ASD05, KL95, KL94] has examined the kinds of messages that relate an action to the goal the action achieves, similar to TraumaTiq's messages about action omission. However, these systems follow the overlay assumption, i.e. the system's knowledge is a superset of the user's; this assumption is not appropriate in the trauma decision support domain. For example, Kosseim and LaPalme use RST to plan instructional text that presents ways for a user to achieve a condition, as in "Turn this knob clockwise and counter-clockwise to minimize interference." This assumption of a naive user who wants to be instructed is different from the approach required in the trauma domain, where the system is trying to persuade a physician to perform an action. In the trauma domain, the physician is the final arbiter of correctness, having more complete knowledge of what is happening in the trauma bay as well as full responsibility for the outcome. The system therefore assumes that the physician *knows* how to achieve a treatment goal, but that the physician either 1) is overlooking a step in achieving the goal and only needs to be reminded of the step; 2) needs to be reminded to *have* the treatment goal; or 3) should be using an alternative method to achieve the treatment goal. Thus while [KL95] use relations such as PURPOSE, RESULT, and CONDITION, *RTPI* must use relations such as Moore's PERSUADE and MOTIVATE [Moo95].

2.4 Introduction to *RTPI*

The generation of multi-sentential discourse has focused on generating text that accomplishes one particular rhetorical goal, such as describing a physical device. Many natural language systems have been developed to generate coherent text plans [MP93,

* Caution: check for medication allergies and order pulmonary care immediately to treat the left pulmonary parenchymal injury.

Figure 2.4: An example of a single message produced by TraumaTiq. This message is about an error of omission.

Hov91, WH96, ZM95]. However, little attention has been given to taking a set of independently generated yet inter-related text plans and producing integrated plans that realize all of the communicative goals in a concise and coherent manner.

To deliver real-time decision support in trauma management, a text generation system must be able to take an arbitrary and often inter-related set of communicative goals and produce a message that realizes the entire set in as concise and coherent a manner as possible. *RTPI* (Rule-based Text Plan Integrator) was designed to perform this task. The need for coherence requires that the system be able to identify and resolve conflict across multiple, independent text plans, and exploit relations between communicative goals. Conciseness requires the ability to aggregate and subsume communicative goals. Although this work was motivated by the need to produce coherent, integrated messages from the individual critiques produced by a decision support system for emergency center trauma care, this same task will arise in future systems as they make use of independent modules that need to communicate with a user. Thus the system should have simple, domain-independent rules, but should also be flexible enough to allow the addition of rules specific to the domain at hand.

2.5 Creating text plans for multiple goals

The input to *RTPI* consists of a *set* of text plans, each containing a communicative goal and the recommended means to achieve that goal in the plan. For example, in Figure 2.4 the communicative goal is to get the physician to treat (or finish treating) the left pulmonary parenchymal injury. The suggested means for accomplishing the treatment

A set of messages produced by TraumaTiq:
⋆ Caution: check for medication allergies and order pulmonary care immediately to treat the left pulmonary parenchymal injury.
⋆ Caution: check for medication allergies and order pulmonary care immediately to treat the compound rib fracture of the left chest.
⋆ Caution: check for medication allergies and do a laparotomy immediately to treat the intra-abdominal injury.
⋆ Caution: do a laparotomy and repair the left diaphragm immediately to treat the lacerated left diaphragm.
⋆ Consider checking for medication allergies now to treat a possible GI tract injury.

RTPI's merged message:
Caution: check for medication allergies as part of treating the left pulmonary parenchymal injury, treating the compound rib fracture of the left chest, treating the intra-abdominal injury, and treating a possible GI tract injury. Then order pulmonary care to complete treating the left pulmonary parenchymal injury and treating the compound rib fracture of the left chest, and do a laparotomy to complete treating the intra-abdominal injury.

Figure 2.5: Original Messages and a Merged Message

goal is to "check for medication allergies and order pulmonary care immediately". When presented with a *set* of text plans for messages, *RTPI* uses a set of transformational rules to transform these into coherent message units that achieve the complete set of goals. These message units are then translated into natural language using templates.

The purpose of *RTPI*'s messages is to support decision-making by the physician. More specifically, since TraumaTiq only generates messages when a critique of the current inferred physician plan is appropriate, the message is intended to persuade the physician to order an action, substitute an alternative action for a previously ordered action, change the order of intended actions, or cancel an existing order about an action. The organization of the final message unit affects how well the message supports this purpose.

The critical, fast-paced nature of trauma management and communication in the

emergency room trauma bay influenced several decisions that affect the message integration process. While messages about actions could be re-organized strictly in the temporal order of the recommended actions, or in order of their potential importance, organizing messages in terms of the relevant treatment goal each action was supporting seemed most likely to be effective in changing physician behavior, since the physician would see the action reference in the context of the reason to perform the action. This would avoid the physician having to deduce the reason the system had for recommending the action. In other words, the system's recommendations should continue to be organized in terms of relevant domain goals, so that the physician can easily evaluate the reasoning behind the system's recommendations and decide whether to adopt them. This decision to organize action messages in terms of goals was confirmed as appropriate by our physician consultant [Cla98].

Another principle that guided rule design in *RTPI* was the reduction of repetition. When different messages contain references to the same action, naming the action multiple times could be construed as suggesting that the action be performed more than once. This misinterpretation is possible because some actions, such as "insert chest tube" *may* be ordered multiple times, and each time *should* indicate a new chest tube. In addition, unnecessary repetition increases message length.

Reducing message length and message quantity (i.e. the number of distinct messages in a message unit) are two more guiding principles of *RTPI*'s rule design. Since the emergency room is a chaotic setting for time-critical decisions, communication must be succinct. A physician's attention is divided over many areas (the patient, other members of the trauma team, various instruments, etc.), so reducing the number of words that a physician must process (whether the words are read or heard) was an obvious design goal if it could be accomplished without sacrificing clarity. This principle guided rule design towards making both the integrated messages and the set of messages concise (i.e. reducing the overall number of messages). Again, while this design decision has not been

tested in a trauma bay due to implementation difficulties not related to this system, this design decision was confirmed by our physician consultant [Cla98].

Finally, since the system's social role on the medical team is that of an expert consultant to the physician who retains ultimate responsibility for the quality of patient care, *RTPI* must recognize that the physician can ignore its recommendations. This differs from other scenarios, such as tutoring, where the system is the sole arbiter of correct behavior. This principle affects the design and application of rules in two ways. It affects how apparent conflict between messages is resolved, as described in Section 2.6.4; and it affects the realization of the text plan trees as described in Section 3.3.3.

Thus *RTPI*'s rules were designed with the following objectives:

1. group actions by relevant treatment goals;
2. avoid repeated mention of the same actions;
3. produce concise messages;
4. produce few, rather than many, individual messages; and
5. reflect the system's social role as a supporter of the physician who has ultimate responsibility for the case.

2.6 Text Plan Transformation Rules in *RTPI*

RTPI's input consists of a set of text plans, each of which has a top-level communicative goal. Rhetorical Structure Theory [MT87] posits that a coherent text plan consists of segments related to one another by rhetorical relations such as MOTIVATION or BACKGROUND. Each text plan presented to *RTPI* is a tree structure in which individual nodes are related by RST-style relations. The top-level communicative goal for each text plan is expressed as an intended effect on the user's mental state [Moo95], such as `(GOAL USER (DO ACTION27))`. The kinds of goals that *RTPI* handles are typical of decision support systems, critiquing systems, systems that provide instructions for performing a

task, etc. These goals may consist of getting the user to perform actions, refrain from performing actions, use an alternate method to achieve a goal, or recognize the temporal constraints on actions.

2.6.1 Algorithm

RTPI operates on RST-style text plan trees. Since TraumaTiq does not produce full text plan trees, an intermediate processing stage converts the logical form of TraumaTiq's simple messages into RST-style trees. A set of these trees is then used as input to *RTPI*. *RTPI* then applies a succession of rules to a set of text plan trees. Each rule specifies the structure of the trees that it operates on, the structure of the tree resulting from application of the rule, and a heuristic which gives a numeric score representing how effective the rule is at achieving *RTPI*'s goals of increased coherence, conciseness, and readability.

Rules are defined in terms of tree specifications and operators, and are stylistically similar to the kinds of rules proposed in [WH96]. When all the tree specifications are matched, the score function of the rule is computed. The score function is a heuristic specific to each rule, and is used to determine which rule instantiation (i.e. which set of text plans to which the rule can be applied) has the best potential text realization. Scores for aggregation rules, for example, measure the opportunity to reduce repetition through aggregation, subsumption, or pronominal reference, and penalize for paragraph complexity.

Once a rule instantiation is chosen by score, the system performs any substitutions, pruning, and moving of branches specified by the rule's operators. The rules currently in use operate on text plan trees in a pairwise fashion, and recursively add more text plans to larger, already integrated plans.

The order of application of the rules is determined by rule type. The four rule types described below are applied in order listed, which is designed to maximize derived benefit:

1. The rules that resolve conflict, due to my hypothesis that the presence of conflict will most seriously hamper assimilation of a message;

2. The aggregation rules which reduce repetition and improve conciseness;

3. Rules that exploit relations between text plans, since they enhance coherence by explicitly connecting different, yet related, communicative goals; and

4. Rules that move messages so that they are presented in proximity to messages about the same topics. Rules that find and associate "trailing comments" are simpler than the other rule types, and so are presented separately in Chapter 3.

Within each rule type group, each rule is tried on pairs of text plan trees within a set of trees representing a message set. Each rule then returns a heuristic score for each tree pair, estimating the benefit it will provide if applied to that pair. The rule instantiation with the highest heuristic score is chosen and the rule's operator is applied to the affected trees. The process is recursively run on the resulting group of trees until no more rules from the current group apply; then the next group of rules is applied.

When the system has applied rules from each of the four types as many times as possible (i.e. no more rules can be applied), or if a preset run time is almost expired, the set of text plans are realized via templates (see Section 3.3).

Since the rules are designed to apply incrementally to a set, every application of a rule results in an improvement in the coherence or conciseness of the tree set, and the tree set is always a viable set of text plans, making the algorithm *anytime* [GL94]. The user can set a time limit for processing of a tree set, and the algorithm can return an improved set at any time. In practice, however, the processing takes less than than one second, even for large (25 plans) input sets, so the time limit is intended to be useful for future, more complex implementations.

```
    ;Do {A} before {B}.
    ;Do {A} before {D}.
    ;=>
    ;Do {A} before {B} and {D}.
(;effects
 ((?tree1 . ((?b . (aggregate 'and ?b ?d))))) ;substitutions
 ((tree-remove ?tree2 *CURRENT-TREES*))       ;post code
 (length ?a)                                  ;score function
 )

(;tree pattern specifications
 (?tree1 . (;partial first tree specification
  ((rootp ?n1))  ;preconditions about pattern
  (?n1 (GOAL USER (KNOW USER (IN-ORDER ?a ?b)))))) ;pattern
 (?tree2 . (;partial second tree specification
  ((rootp ?n2)) ;preconditions about pattern
  (?n2 (GOAL USER (KNOW USER (IN-ORDER ?a ?d)))))) ;pattern
)
```

Figure 2.6: A rule for aggregating temporal constraints.

2.6.2 The form of the rules

Each rule specification consists of the following structure. Names that begin with question marks, as in "`?x`", represent variables that unify with portions of a text plan. Another kind of variable used in some specifications, "`&x`", will be explained when it appears later. A specification consists of:

- rule name
- effects section:
  - substitutions: Using the names matched in the "pattern" section, replace, in the specified tree, the first item with the second. In Figure 2.6, the substitution list says to replace whatever matched `?b` in `?tree1` with the result of `(aggregate 'and ?b ?d)`

- other instructions (in Lisp): in Figure 2.6, since the goals of `?tree2` are now incorporated into `?tree1` because of the substitution above, remove `?tree2` from the current set of text plan trees.
- score function: In Figure 2.6, the score function is simply the length of the list of actions bound to the variable `?a`. Intuitively, the rule eliminates one mention of each action in the list, so that is the rule's degree of goodness.

- List of tree pattern specifications
  - preconditions: predicate statements about variables that must hold for the rule to apply. For `?tree1` in Figure 2.6, the predicate states that whatever node is bound to `?n1` must be the root of a tree. The precondition prevents the rule from applying to a node of this type that has a tree structure above it, perhaps because it has already been incorporated into a tree by another rule application.
  - pattern: Two tree patterns are listed in Figure 2.6, along with a description of nodes that must appear in the tree. This specification for `?tree1` says roughly "This tree must have a node, bound to `n1`, that contains the predicate `(GOAL USER (KNOW USER (IN-ORDER ?a ?b)))`", where `?a` and `?b` are variables that can match whatever is inside the predicate. Here they will each match a list of one or more actions. However, the variable `?a` must be bound to the same value as the `?a` in the second tree specification.

Figure 2.6 illustrates a very simple example rule[3]. A unifier matches the RST-style tree specification for tree `?tree1` from the pattern specifications of the rule as follows. First, it checks a table of nodes to find a node that matches the pattern for `?n1`. Once a match for `(GOAL USER (KNOW USER (IN-ORDER ?a ?b)))` is

[3] Slight modifications have been made to the rule forms presented to make them less busy and easier to understand.

located, the unifier checks any other patterns specified for ?tree1; in this case there are no other specifications. If all node patterns have matches within tree ?tree1, the system then checks the list of preconditions. For ?tree1 the only precondition is that the predicate "rootp" hold true for whatever has unified with ?n1. In other words, the node in ?tree1 that now corresponds to ?n1 must be the root of the tree for this rule to apply.

Once the whole tree specification for ?tree1 is matched, the system attempts to find a match for the remaining tree specifications (here, only ?tree2). If trees that meet the specifications for ?tree1 and ?tree2 are both found, then the score function can be calculated using the instantiations of the variables. In Figure 2.6, the score function represents the improvement in the final message expected from having to say the items in ?a only once (?a may be a list), whereas ?a occurs twice in the original text plans. Thus the score function for this rule is simply the length of the list of actions unifying with ?a.

When the system applies a rule, the *effects* section of the rule specifies what to do with (or to) the two trees; in this case, all instances of ?b in tree ?tree1 are replaced with the aggregation of the instantiations of ?b and ?d. Then it executes any instructions in the "other instructions", which here say to remove tree ?tree2 from the plan set since its communicative goal is now achieved by the altered ?tree1.

2.6.3 Classes of Rules

RTPI has three main classes of rules, all of which produce an integrated text plan from separate text plans. The classes of rules correlate with the three categories of problems that I identified from the analysis of TraumaTiq's messages, namely, the need to: 1) resolve apparent conflict among text plans (Section 2.6.4); 2) aggregate communicative goals to achieve more succinct text plans (Section 2.6.5); and 3) exploit the relationships between communicative goals to enhance coherence (Section 2.6.6). In addition, simpler rules re-order related messages so they may be presented together, as described in Section 3.2.

2.6.4 Resolving Conflict

The ability to recognize and resolve conflict is required in a text planner because *both* the appearance and the resolution of conflict can be the result of text structure. *RTPI* identifies and resolves a class of domain-independent conflict, and uses a resolution strategy dependent upon the social relationship between the user and the system.

One class of conflict that can best be resolved at the text planning level results from implicit messages in text. Resolving conflict of this kind within independent modules of a critiquing system would require sharing extensive knowledge, thereby violating modularity concepts and making the planning process much more complex. For example, suppose that the user has conveyed an intention to achieve a particular objective by performing act A_u. One system module might post the communicative goal of getting the user to recognize that act A_p must precede A_u, while a different module posts the goal of getting the user to achieve the objective by executing A_s instead of A_u. While each of these communicative goals might be well-motivated and coherent in isolation, together they are incoherent, since the first presumes that A_u *will* be executed, while the second recommends retracting the intention to perform A_u. A text planner with access to both of these top-level communicative goals *and their text plans* can recognize this implicit conflict and revise and integrate the text plans to resolve it.

There are many ways to unambiguously resolve this class of implicit conflict. Strategy selection depends on the *social relationship* between the system and the user. This relationship is defined by the relative levels of knowledge, expertise, and responsibility of the system and user. Three possible strategies and their motivations are:

1. Discard communicative goals that implicitly conflict with a system recommendation. In the above example, this would result in a text plan that only recommends doing A_s instead of A_u. This strategy would be appropriate if the system is an expert in the domain, has full knowledge of the current situation, and is the sole

TraumaTiq messages:
Performing local visual exploration of all abdominal wounds is preferred over doing a peritoneal lavage for ruling out a suspicious abdominal wall injury.

Please remember to check for laparotomy scars before you do a peritoneal lavage.

Message from *RTPI* integrated plan:
Performing local visual exploration of all abdominal wounds is preferred over doing a peritoneal lavage for ruling out a suspicious abdominal wall injury. However, if you do a peritoneal lavage, then remember to first check for laparotomy scars.

Figure 2.7: An example of a rule application removing the appearance of conflict. The rule is shown in Figure 2.8.

---

arbiter of correct performance. An instructional or tutoring system might be in this relationship to a user.

2. Integrate the text plan that implicitly conflicts with the system recommendation as a concession that the user may choose not to accept the recommendation. This strategy is appropriate if the system is an expert in the domain, but the user has better knowledge of the current situation and/or retains responsibility for selecting the best plan of action. Decision support is such an environment. The top half of Figure 2.7 presents two TraumaTiq messages that exhibit implicit conflict, while the bottom part presents the English realization of *RTPI*'s integrated text plan, which uses a CONCESSION relation to achieve coherence.

3. Present the system recommendation as an alternative to the user plan. This may be appropriate in a domain where the user has more complete knowledge and more expertise.

```
(conflict-resolution-02
 ;Do A instead of B because C. However, if you do B, do D first.
 (;effects
  ()        ;substitution list
  ((push (make-concession-tree ?tree1 ?tree2)  ;post-code
     *CURRENT-TREES*)
   (t-remove ?tree1 *CURRENT-TREES*)
   (t-remove ?tree2 *CURRENT-TREES*))
  1 ;score function
  )

 (;tree pattern specifications
  (?tree1 . (
         ((rootp ?n1))
         (?n1 (GOAL USER ( KNOW USER ( PREFERRED ?a ?b ?c))))))
  (?tree2 . (
         ((rootp ?n2))
         (?n2 (GOAL USER ( KNOW USER ( IN-ORDER ?d ?b))))
      )))
 )
```

Figure 2.8: A rule for resolving apparent conflict between two messages. The result of applying this rule can be seen in Figure 2.7.

Clearly the second strategy is most appropriate in TraumAid's domain, where the system is indeed an expert, but with its knowledge reliant on data entry, whereas a physician is aware of the entered data in addition to the wealth of data available by direct observation. Furthermore, the physician bears sole responsibility for the outcome of the case. The appropriateness of this choice for *RTPI*'s rule design was confirmed by our emergency room physician consultant [Cla98]. It is important for the resulting message to be framed in light of this social role of the system, i.e. as an expert advisor. Implementing the second strategy requires the rule to integrate the messages with a CONCESSION relation, which provides a plan structure enabling realization to text that embodies the strategy, as discussed in Section 3.3.

A typical conflict resolution rule is presented in Figure 2.8. This rule first matches

a message that draws the physician's attention to a procedure that should not be performed, binding it to `?tree1`. The predicate PREFERRED has three arguments to match its use in the current domain, where preference expresses a preferred action `?a`, a sub-optimal action to be replaced, `?b`, and (optionally) a reason for the preference `?c`. Next the rule finds a binding for a scheduling constraint on `?b` that indicates some action, in this case `?d`, should precede the performance of `?b` (the sub-optimal action). There is no other use of `?d` in the rule because it does not matter what is bound to `?d`; the text resulting from the rule will work regardless of what is bound to `?d`: "If you do `?b`, be sure to do `?d` first." The effects code of the rule then superimposes a CONCESSION relation above the two trees. The score for this rule is always `1` in the TraumAid domain, since the rule does not allow any mentions of actions or goals to be avoided, and TraumaTiq's design does not permit multiple messages of these types to be issued about a single action. In other words, no instantiation of this rule will ever be in competition with another. If this rule were to be used in a different domain a dynamic scoring function might need to be designed.

The existence of apparent incoherence, which is potentially as serious as apparent conflict, is addressed by [HG05], but their system is generating several sentences directly from a single knowledge set. They note that indiscriminate opportunistic use of modifiers can result in the apparent existence of two princesses:

The pretty princess lived in the castle. The blonde princess loved a knight.

when only one pretty, blonde princess exists in the knowledge base. However, *RTPI* is unique in addressing apparent conflict that is introduced when messages from two coherent sources (in this case separate modules of TraumaTiq) are juxtaposed[4].

[4] The reason that two modules of TraumaTiq can generate messages that appear to be in conflict is that one module is generating information about preconditions based on the physician's planned actions (since the physician may choose to continue this plan, it is important that precondition information be shared), while another module is generating messages based on TraumAid's plan.

TraumaTiq messages:
** Caution: check for medication allergies and do a laparotomy immediately to treat the intra-abdominal injury.*
** Consider checking for medication allergies now to treat a possible GI tract injury.*
** Please remember to check for medication allergies before you give antibiotics.*

Message from *RTPI* integrated plan:
Caution: check for medication allergies to treat the intra-abdominal injury and a possible GI tract injury, and do it before giving antibiotics. Then do a laparotomy to complete treating the intra-abdominal injury.

Figure 2.9: Result of communicative goal aggregation.

2.6.5 Aggregation Through Plan Integration

Our analysis of TraumaTiq's output showed that one prevalent problem was informational overlap, i.e. the same actions and objectives often appeared as part of several different input text plans, and thus the resulting messages appear repetitious. Aggregation of the communicative goals associated with these actions and objectives allows *RTPI* to make the message more concise. Figure 2.9 illustrates a set of messages that involve repetition of suggested actions, and the lower part shows output text which results from the communicative goal aggregation.

Aggregation of overlapping communicative goals is not usually straightforward, however, and often requires substantial reorganizing of the trees. *RTPI*'s approach is to draw on the ordered, multi-nuclear SEQUENCE relation of RST. Separate plans with overlapping communicative goals can often be reorganized as a sequence of communicative goals in a single plan. The recommended actions can be distributed over the sequentially related goals *as long as* the new plan still captures the relationships between the actions and their motivations given in the original plans.

Original TraumaTiq messages:
Caution: check for medication allergies, give antibiotics, and do a laparotomy immediately to treat the intra-abdominal injury.
Caution: check for medication allergies, give antibiotics, and inspect the duodenum immediately to detect the possibility of a duodenal injury.

Integrated *RTPI* message:
Caution: check for medication allergies and give antibiotics to treat the intra-abdominal injury and detect the possibility of a duodenal injury. Then do a laparotomy to complete treating the intra-abdominal injury, and inspect the duodenum to complete detecting the possibility of a duodenal injury.

Figure 2.10: Example of the application of a complex aggregation rule to two messages (see the rule in Figure 2.11), where the first segment is empty. In this example from TraumAid case data, additional messages were integrated before the system was finished processing the message set.

2.6.5.1 Integrating text plans with sequenced actions

For example, one complex class of aggregation is the integration of text plans that have overlapping actions or objectives, but which also contain actions and objectives that do not overlap. When those that overlap can be placed together as part of a valid sequence, a multi-part message can be generated. In Figure 2.10, the actions "check for medication allergies" and "give antibiotics" are present in both original messages, resulting in repeated references to the actions. An *intersection* is a set of such messages which share actions, and so are combined into one larger message with fewer references to the shared actions. An *omission error* is reported by TraumaTiq when the physician plan does not include a procedure that TraumAid's plan includes. The top level communicative goal of a plan tree for an error of omission message contains a goal to get the user to perform the omitted action, e.g. `(GOAL USER (DO USER Laparotomy))`. Intersections of omission errors are formed when two or more messages about omissions share consecutive actions, subject to the acceptability of the score calculated by the rule used to create the intersection. The

```
(complex-aggregation-01
        ;Do A1, A2 to address B.
        ;Do A2, A3 to address C.
        ;=>
        ;Do A1 to address B. Next do A2 to address B and C.
        ;Then do A3 to finish C.

        ;A2 is the list of consecutive shared actions that
        ;address goal set B.
 (;effects
  ()                                    ;substitution list
  (                                     ;post-code
   (push (make-aggregate-intentions-tree
            ?n2 ?n4 (merge %a %a') () ?tree1 ?tree2)
            *CURRENT-TREES*)
   (t-remove ?tree1 *CURRENT-TREES*)
   (t-remove ?tree2 *CURRENT-TREES*)
   )
   ;score function
  (+ 1 (* 2 (length (second (merge %a %a'))))
       (- (length (third (merge %a %a'))))
       (- (if (and (first (merge %a %a'))
                   (third (merge %a %a'))) .5 0)))
 )

 (;tree pattern specifications
  (?tree1 . (
              ((rootp ?n1))             ;preconditions about tree
              (?n1 (goal USER ( DO USER %a)))
              (?n2 (BEL USER (?relation %a ?goals1)))))
  (?tree2 . (
              ((rootp ?n3))
              (?n3 (goal USER ( DO USER %a')))
              (?n4 (BEL USER (?relation %a' ?goals2))))))
)
```

Figure 2.11: A rule for integrating messages that share actions which have partial order constraints. The rule generates a message containing up to three conceptual segments.

---

benefits and drawbacks of an intersection versus the original messages are weighed in the scoring function associated with each rule.

An example of a rule that generates such a message from multiple omission error messages is shown in Figure 2.11. This rule produces an integrated text plan comprised of up to three sequentially related segments (referred to as the *pre-list, mid-list,*

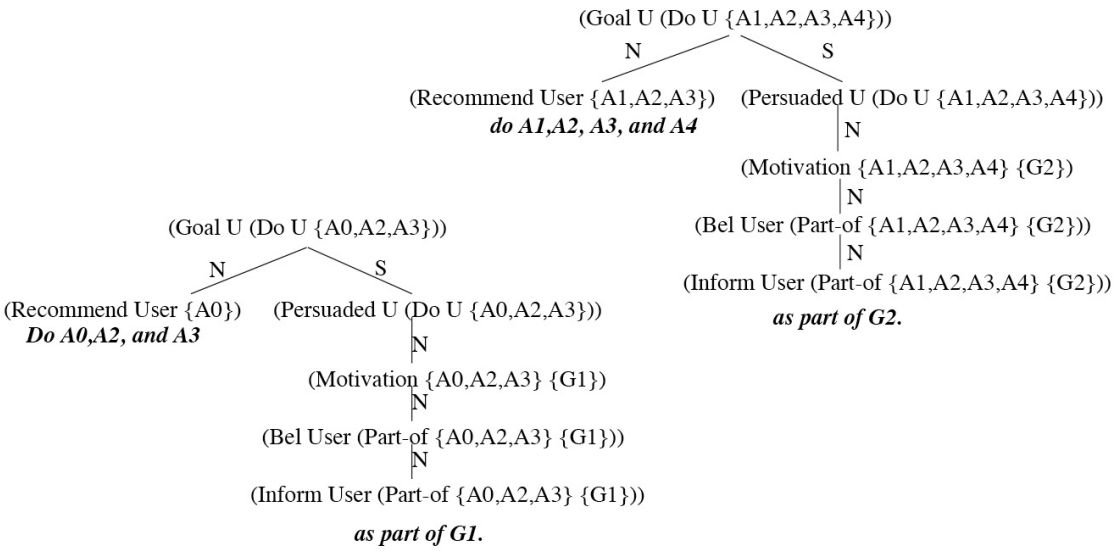


Figure 2.12: Two text plans that require a complex aggregation rule. The resulting integrated text plan is shown in Figure 2.15.

---

and *post-list*), with the middle segment conveying the shared actions and their collected motivations. The pre-list and post-list segments convey the actions that temporally precede or follow the shared actions, and are also presented with their motivations. In this rule, the "merge" of `%a` and `%a'` is a three part list representation of *all actions from both messages* to which the rule is applied. Within the representation, the three action lists correspond to the pre-list, mid-list, and post-list of the intersection produced by the rule. The mid-list contains the longest temporally consecutive set of shared actions from the two messages. The pre-list then contains any actions that precede actions in the shared mid-list, and the post-list contains the remaining actions (which must follow the shared actions). For example, in Figure 2.10 the pre-list is empty, the mid-list is the list `(check_for_medication_allergies, give_antibiotics)` , and the post-list is the list `(do_a_laparotomy, inspect_the_duodenum)` . The presence of the single variable `?relation` in the specifications for both trees requires that the actions in both trees

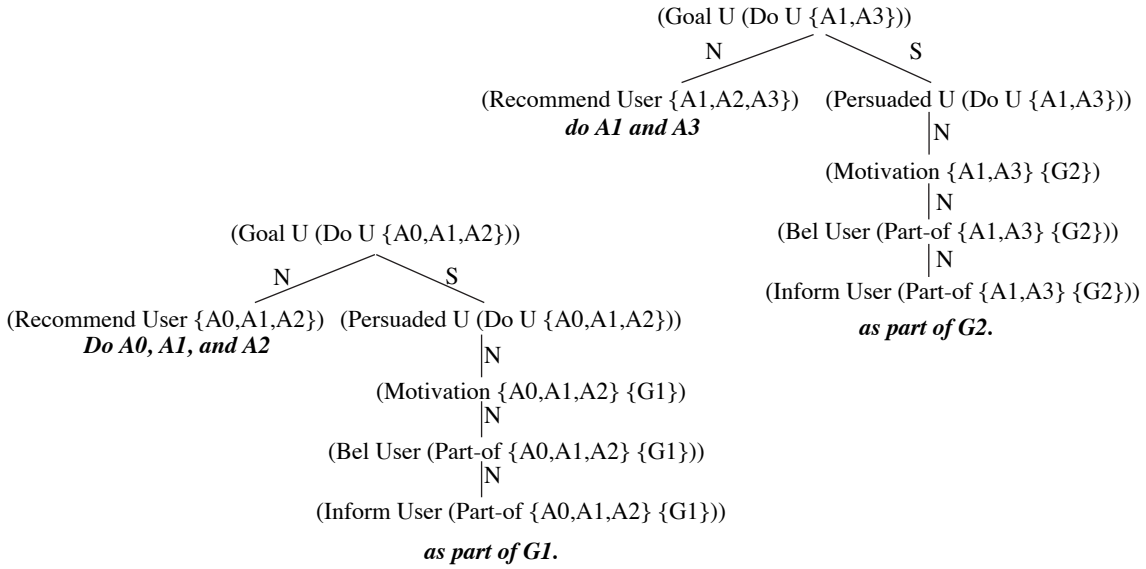


Figure 2.13: Text plans for two messages from one of TraumAid's example cases. The resulting *RTPI* integrated text plan is shown in Figure 2.16.

---

have the same relationship to the goals.

For example, suppose that one text plan has the goal of getting the user to perform actions $A0, A2,$ and $A3$ to achieve $G1$, while a second text plan has a goal of getting the user to perform $A1, A2, A3,$ and $A4$ to achieve $G2$, as shown in Figure 2.12. Then in the rule shown in Figure 2.11, the result of `(merge %a %a')` will be the structure $\{\{A0, A1\}, \{A2, A3\}, \{A4\}\}$, so that the mid-list contains the shared actions $A2$ and $A3$. Figure 2.15 on page 40 presents the text plan resulting from the application of this rule. The rule's score function for a tree with non-empty pre-, mid-, and post-lists is

$$1 + (2 * length(mid)) - length(post) - 0.5$$

Thus this instantiation of the rule has a score of $1 + (2 * 2) - 1 + -0.5$, or 3.5. Because the

Original TraumaTiq messages:
Caution: transport the patient to the x-ray department, check for medication allergies, and get a CT-scan of the abdomen immediately to rule out a compound fracture of a lumbar vertebra.
Caution: check for medication allergies and do a laparotomy immediately as part of treating the intra-abdominal injury.

Integrated *RTPI* message:
Caution: transport the patient to the x-ray department as part of ruling out a compound fracture of a lumbar vertebra. Next check for medication allergies to address this goal and also to treat the intra-abdominal injury. Then get a CT-scan of the abdomen to complete ruling out a compound fracture of a lumbar vertebra, and do a laparotomy to complete treating the intra-abdominal injury.

Figure 2.14: Example of the application of a complex aggregation rule to two messages (see the rule in Figure 2.11), where all three segments have data. In this example from TraumAid case data, additional messages were integrated before the system was finished processing the message set.

---

rule's score is higher than zero, it will fire as long as another pair of trees does not yield a higher score with the same rule. The subtraction of 0.5 is to reflect my hypothesis that trees with both a pre- and post-list should be penalized for their complexity, relative to trees that have only two parts. Realization of this text plan in English (see Section 3.3.5 for details) produces the message:

> *Do* $A0$ *as part of* $G1$*, and* $A1$ *as part of* $G2$*. Next do* $A2$ *and* $A3$ *to address both of these goals. Then do* $A4$ *to complete* $G2$*.*

This kind of aggregation is not possible in systems that apply unrestricted clause re-ordering in order to enable aggregation (e.g. systems such as Sentence Planner or REVISOR). Such systems cannot ensure that partial ordering restrictions will be maintained when clauses are re-organized. *RTPI* can aggregate these messages in the presence of partial ordering restrictions because it exploits these restrictions in its rules.

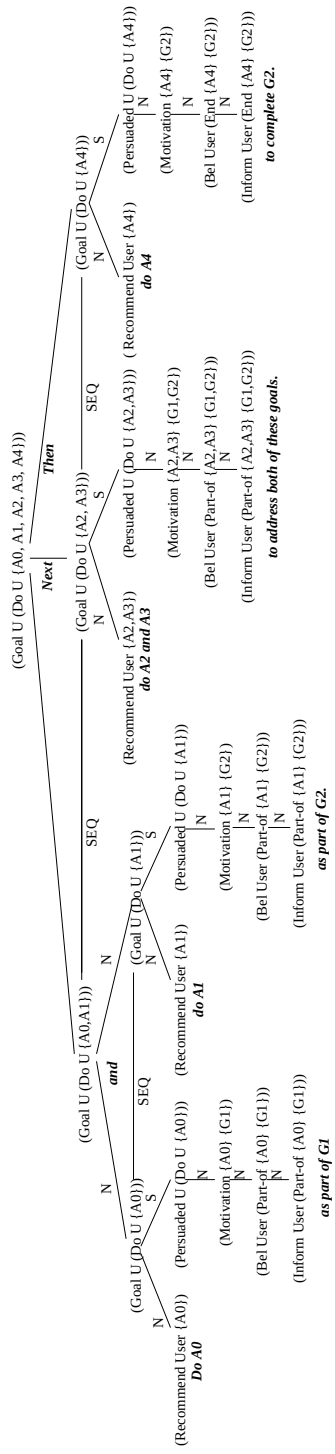


Figure 2.15: Result of a complex aggregation rule applied to the text plans in Figure 2.12.

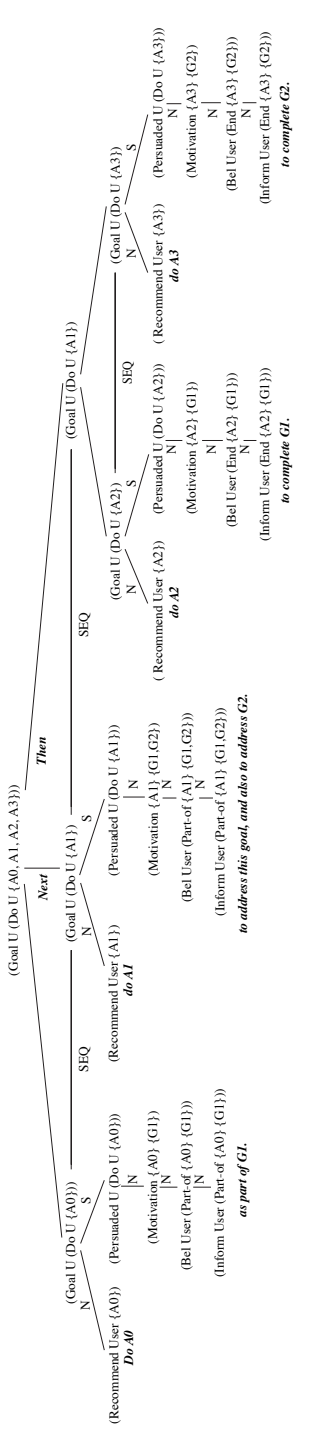


Figure 2.16: Result of a complex aggregation rule applied to the text plans in Figure 2.13.

Two different applications of this complex aggregation rule to actual data emergency room data from TraumAid and TraumaTiq are shown in Figures 2.10 and 2.14. In both cases, the message shown is the result of one application of the aggregation rule shown in Figure 2.11, and in both cases additional messages were added to the integrated message before it was presented by *RTPI*. Figure 2.14 shows the message that corresponds to the tree shown in Figure 2.16, which is the result of integrating the plan trees in Figure 2.13. In this example, the pre-list consists of $A0$, the mid-list of $A1$, and the post-list of $A2, A3$. Thus this use of the rule has a score of $1 + (2 * 1) + -2 + -0.5$, or 0.5. Note that if either of these messages had another action in the post-list, then the additional complexity would cause the heuristic score to be below zero and the rule would not fire. Thus this example shows a borderline case, where the improvement derived by re-organizing the message to reduce repetition is almost outweighed by the complexity introduced. Fine tuning of rules' score functions will have to be performed when *RTPI* is fully implemented in a trauma bay, since small differences in phrasing, emphasis, graphics, or other implementation-dependent details could favor one text plan design over the other.

The system is restricted to performing aggregations where all overlapping actions are consecutive. As an example:

```
Message 1: action1 action2 action3 action4 -> goal1
Message 2:         action2         action4 -> goal2
```

cannot be combined because it is too complex for text to point out that two actions are shared (actions 2 and 4), but that there is an intervening action (action 3) which must be performed. This type of overlap was excluded from consideration for integration after reading the text produced by hand from hypothetical cases. The exclusion turned out not to matter in this domain, since this kind of intervening action never appeared in the TraumAid test data.

A rule for forming three segment messages can still fire when the pre- and/or post-lists are empty, but will not fire if the mid-list is empty. The score increases with the size

Original TraumaTiq messages:
Please remember to check for medication allergies before you give antibiotics.
Please remember to check for medication allergies before you get an IVP.

Integrated *RTPI* message:
Please remember to check for medication allergies before you give antibiotics and get an IVP.

Figure 2.17: Simple aggregation of text plan trees with the same structure.

---

of the mid-list, since the list reduces the number of times an action must be mentioned. However, the score is reduced according to the size of the post-list (since the post-list requires additional mentions of the motivations), and is also reduced if both the pre- and post-list are non-empty (since that increases the complexity of the paragraph, requires more mentions of the motivations, and therefore may reduce the benefit of applying the rule).

RTPI rules emphasize organization by objectives (appropriate in the trauma decision-support environment [Cla98]). In this application an arbitrary limit of three is placed on the number of sequentially related segments in a multi-part message, though each segment can still address multiple goals. The design decision to limit the integrated message to three segments was based on the observation that four or more segments reduced the degree to which the message was organized around treatment goals, making the message appear to be an action recipe; and that four or more segments made it very difficult to realize any gains through use of referring phrases without introducing confusion. Three segment messages allow the reorganization of communicative goals to enable aggregation while maintaining focus on objectives. This limit is sufficient for the input from TraumaTiq, but may need to be extended when the rule is used for other applications.

```
(scheduling-same
 ;General scheduling aggregation with same scheduling type only
 ;Do A before B (because <reason>).
 ;Do A before D (because <reason>).
 ;=>
 ;Do A before B and D (because <reason>).
 (;effects
  ( ;substitution list
   (?tree1 . ((?b . (aggregate ?b ?d))))
   )
  ((t-remove ?tree2 *CURRENT-TREES*))   ;post-code
  1                                     ;score function
  )

 (;tree pattern specifications
  (?tree1 . (
      ((rootp ?n1)) ;preconditions about tree
      (?n1 (GOAL USER (KNOW USER (IN-ORDER ?a ?b))))
      (?n2 (EVIDENCE USER (IN-ORDER ?a ?b) ?reason))
      ))
  (?tree2 . (
      ((rootp ?n3))
      (?n3 (GOAL USER (KNOW USER (IN-ORDER ?a ?d))))
      (?n4 (EVIDENCE USER (IN-ORDER ?a ?d) ?reason))
      )))
 )
```

Figure 2.18: Simple integration rule for two trees of the same type.

2.6.5.2 Aggregation across plans with similar communicative goals

The integration described in Section 2.6.5.1 is performed on one or more text plans for errors of omission, all of which share the communicative goal of getting the physician to perform an action missing from the physician's plan. In addition to these complex aggregation rules for sequences of actions, there are rules for integrating other kinds of text plans that have similar communicative goals. While TraumaTiq will report multiple actions that relate to a single treatment goal in a single critique, resulting in messages of the form "Do A, B, and C to achieve goal D", scheduling critiques do not have treatment goals. Instead they relate two actions, and TraumaTiq does not combine acts in scheduling messages. TraumaTiq has six kinds of scheduling messages (see Appendix A) in which

Original TraumaTiq messages:
Caution: get a chest x-ray immediately to evaluate the chest.

Please get a chest x-ray before getting a urinalysis because it has a higher priority.

Integrated *RTPI* message:
Caution: get a chest x-ray to evaluate the chest, and do it before getting the urinalysis because it is a higher priority.

Figure 2.19: A message about action omission is integrated with a scheduling message that has the same action as its subject.

---

two actions are presented as having a preferred order.

RTPI combines scheduling messages that share actions, and frequently the two messages represent the same type of scheduling information. The first kind of combination has two plan trees of the same type sharing one action that should precede others. For example, Figure 2.17 shows two scheduling messages that place a temporal constraint on their first action "check for medication allergies". The integration rule is shown in Figure 2.18. The tree specification has an EVIDENCE predicate with a variable that binds to the reason that the actions should be performed in order. In this example the reason in both text plans is that "check for medication allergies" is a PRECONDITION for both giving antibiotics and getting an IVP. The reason is not explicitly given in the message text because the TraumaTiq system assumes the physician is aware of this relation, but is present in the plan tree.

The second kind of combination is the case where the same action (here action6) appears in the second slot in two of the same type of scheduling message:

action1 before action6

action2 before action6

→ action1 and action2 before action6

TraumaTiq messages:
** Caution: check for medication allergies and do a laparotomy immediately to treat the intra-abdominal injury.*
** Consider checking for medication allergies now to treat a possible GI tract injury.*
** Please remember to check for medication allergies before you give antibiotics.*

Message from *RTPI* integrated plan:
Caution: check for medication allergies to treat the intra-abdominal injury and a possible GI tract injury, and do it before giving antibiotics. Then do a laparotomy to complete treating the intra-abdominal injury.

Figure 2.20: Result of communicative goal aggregation, reproduced from Figure 2.9.

The precedence between medical acts for the trauma domain is consistent [Cla98] and is established in a database, so that `action1` and `action2` can always be mentioned in the correct order.

2.6.5.3 Aggregation across plans with different communicative goals

RTPI can also handle aggregation when actions or objectives are shared between different kinds of communicative goals. For example, scheduling messages may combine with omission messages if the actions of the scheduling message are the same as the acts of the omission messages, as in Figure 2.19. They may also combine if the scheduling message contains a subset of the acts of the omission message, producing a result such as:

> *Caution: check for medication allergies, give antibiotics, and order pulmonary care to treat the compound rib fracture of the left chest. Remember to check for medication allergies before you get the IVP.*

Scheduling messages may also combine with multiple omission messages if the arguments of the scheduling message are equal to the arguments of either the pre-list, mid-list, or post-list of the combined omission messages. Consider the example in Figure 2.20, and the text plan trees for this example in Figure 2.21. The first step in this example is

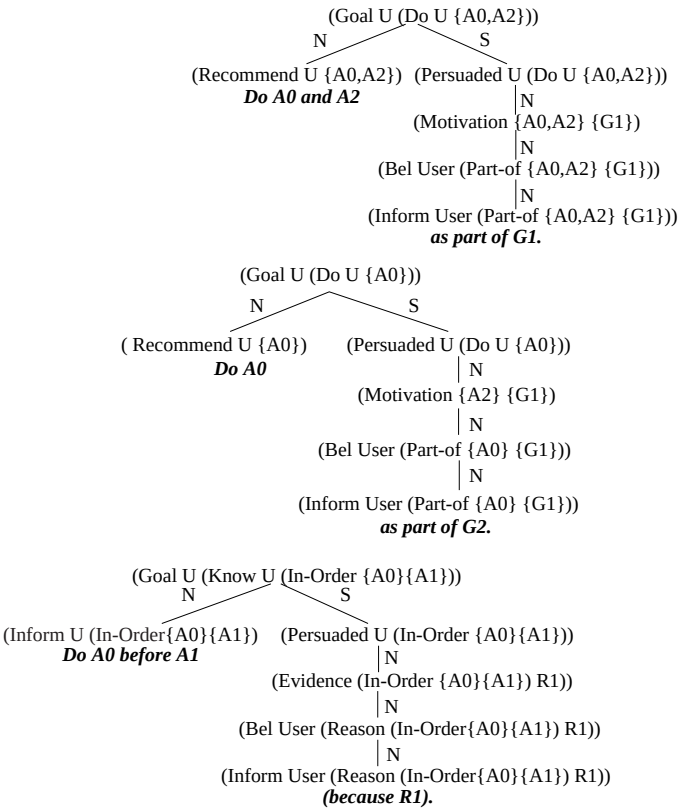


Figure 2.21: Input to *RTPI* (see Figure 2.23).

---

the creation of a combined-goal-sequence tree, using the rule from Figure 2.11, to integrate the first two trees shown in Figure 2.21 by aggregating the communicative goal `(GOAL USER (DO USER A0:check_med_allergies))` [5] that exists in both text plans. Once those two trees are integrated, the rule shown in Figure 2.22 can be applied to add the third tree to the structure. The rule in Figure 2.22 looks for overlap between the communicative goal of getting the user to do a series of actions (from previously integrated trees) and the goal of having the user recognize a temporal constraint on some of the actions. In this rule specification the expression `(second %a)` refers to the mid-list of the structure bound to `%a` (similar rules exist which can attach scheduling information to the pre- and

[5] In this paragraph action labels from Figure 2.21 are concatenated with action names from Figure 2.20 for clarity.

```
(multi-part-agg-sched
 ;Append scheduling messages to combined-goal-sequence tree,
 ;this rule applies to middle segment of sequence only.
 ;Use of %a guarantees middle segment exists.
 (;effects
  ()                                  ;empty substitution list
  ((append-sched-to-sub-tree ?n2 ?tree1 ?tree2)     ;post-code
   (t-remove ?tree2 *CURRENT-TREES*)
   )
  (length (second %a)))                ;score function

 (;tree pattern specifications
  (?tree1 . (
      ((or (null (first %a)) (null (third %a))))
      (?n1 (COMBINED-GOAL-SEQUENCE %a))
      (?n2 (MOTIVATION  %a ?goal ))
      (?n3 (BEL USER (?relation (second %a) ?goal)))))
  (?tree2 . (
      ((rootp ?n4))
      (?n4 (GOAL USER (KNOW USER (IN-ORDER (second %a) ?d)))))))
 )
```

Figure 2.22: This rule integrates two different kinds of messages: a previously integrated message about actions that have not yet been ordered by the physician (?tree1, omission errors) and a reminder message about temporal constraints on those actions (?tree2).

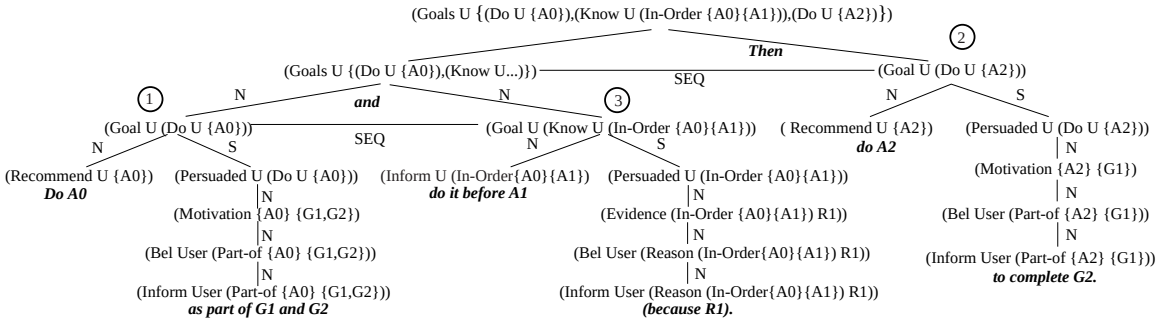


Figure 2.23: Result of two rules applied to input shown in Fig. 2.21. First, a complex aggregation rule integrates two trees from Fig. 2.21 to make a tree with the two subtrees labeled (1) and (2). Next, a rule that places scheduling trees with related goals inserts a third subtree (3).

Original TraumaTiq messages:
* Caution: do a peritoneal lavage immediately as part of ruling out abdominal bleeding.

* Do not reassess the patient in 6 to 24 hours until after doing a peritoneal lavage. The outcome of the latter may affect the need to do the former.

RTPI integrated message:
Do a peritoneal lavage immediately as part of ruling out abdominal bleeding. Use the results of the peritoneal lavage to decide whether to reassess the patient in 6 to 24 hours.

Figure 2.24: Two inter-dependent messages from TraumaTiq, and the integrated message that exploits the relationship between their communicative goals by making the relationship explicit. This example was produced by the rule in Figure 2.25.

---

post-lists). The third tree in Figure 2.21 is present because `A1:give_antibiotics` is already in the physician plan, but `A0:check_med_allergies` is not. In this example the mid-list of the combined-goal-sequence tree consists of `(A0:check_med_allergies)`, and the rule associates the temporal constraint information with the action(s) in the mid-list. The resulting integrated text plan is shown in Figure 2.23.

For additional implementation information on scheduling message integration, see Appendix A.

2.6.6 Exploiting Related Goals

Occasionally two text plans may exhibit no conflict, yet the relationships between their communicative goals can be exploited to produce more coherent text. This kind of plan integration is more complex than the aggregations shown in the previous two sections. For example, consider the two individual messages produced by TraumaTiq shown in Figure 2.24. While the two messages do not conflict, *RTPI*'s rules exploit the relation between the communicative goals in their respective text plans to produce a more concise and coherent message. In particular, the rule shown in Figure 2.25 recognizes the

```
(omit-dependency
   ;omission and scheduling-depend aggregation
   ;Do A to achieve C.
   ;Don't do B until after A. A's outcome may affect the need to do
   ;=>
   ;Do A to achieve C, and use the results of A to decide whether to
   ;perform B.
   (;effects
    ()                                    ;empty substitution list
    (                                     ;post-code
     (push (make-sequence-tree (list ?tree1 ?tree2) 'os-sequence)
           *CURRENT-TREES*)
     (t-remove ?tree1 *CURRENT-TREES*)
     (t-remove ?tree2 *CURRENT-TREES*))
    1 ;score function
    )

   (;tree pattern specifications
    (?tree1 . (;omission
               ((rootp ?n1))
               (?n1 (GOAL USER ( DO USER ?a)))))
    (?tree2 . (;scheduling dependency
               ((rootp ?n2))
               (?n2 (GOAL USER ( KNOW USER ( IN-ORDER ?a ?b))))
               (?n3 (EVIDENCE USER ( IN-ORDER ?a ?b)
                           (DECISION-DEPENDENCY ?b (RESULT ?a)))))
   )
```

Figure 2.25: A rule that exploits the relationship between an omission error and a scheduling dependency. This rule produced the example in Figure 2.24.

---

interaction between an initial plan to get the user to perform an action A_s, and a second plan that gets the user to recognize a temporal dependency between A_s and another action. The first specification is for a tree that persuades the user to do some action `?a`. The second specification says roughly "To achieve the goal of getting the user to know (and believe) that `?a` and `?b` must be performed in order, explain to the user that the decision to perform `?b` is dependent on the information that will be gained by performing `?a`." This rule creates a text plan that results in the message shown in the lower half of Figure 2.24. The new text makes the relationship between the performance of the lavage, on the one hand, and the need to use the results of the lavage, on the other, explicit.

2.7 Summary

Integration of multiple text plans is a task that will become increasingly necessary as independent modules of sophisticated systems are required to communicate with a user. *RTPI* is a rule-based system that draws on rhetorical structure and discourse theory to produce integrated messages from individual ones, each of which is designed to achieve its own communicative goal.

Messages about errors of omission and commission, scheduling, and procedure choice will be common to other systems that support a user in an environment where sets of actions are motivated by certain goals. While only applied within the trauma decision support domain, *RTPI*'s rules were designed to handle general cases of the kinds of messages produced by TraumaTiq, not simply the exact messages shown here. The design of rules for plan integration performed between specific categories of messages will be a useful resource for other system developers in decision support.

RTPI's algorithm applies ordered sets of rules to a set of text plans. A score function associated with each rule determines the plan trees to which the rule is applied. After each rule application the text plans are a viable set of messages; improvement continues until no rules are applicable or no rule application has an acceptable score.

This chapter introduced the method by which *RTPI* integrates text plan trees to form more complex text plans. Chapter 3 will present the last category of rules in *RTPI*, the rules that re-order other related messages so they may be presented together. The chapter also explains the means for realizing *RTPI*'s plans as English text, and presents the results of an evaluation of *RTPI*'s output.

Chapter 3

TRAILING COMMENTS, REALIZATION, AND EVALUATION

3.1 Introduction

The previous chapter introduced *RTPI*'s rule-based text plan integration. Section 3.2 will present the final class of rules in *RTPI*, which enable re-ordering of messages to place related messages in close proximity. Section 3.3 gives examples demonstrating the need for careful realization of integrated text plans, and explains more features of *RTPI*'s realization method as they apply to text plans in general, as well as to specific kinds of text plans. Section 3.4 describes the results of two evaluations of the output of *RTPI*: a simple numeric comparison of various message attributes before and after *RTPI* operates on actual message sets, and an evaluation by human subjects of the integrated messages produced by *RTPI*. The chapter concludes with a summary in Section 3.5.

3.2 Trailing Comments

Trailing comments are messages that are related to previous messages in a set, but not closely enough to allow them to be fully integrated with those messages. Thus while they are found and manipulated by rules similar to those in the previous chapter, the rules are coarser in their operation, simply re-ordering messages where necessary. Also, there are many realization issues surrounding the expression of trailing comments, and so they are presented in this chapter with other realization information.

Occasionally when several text plans ($T_1, T_2, ...$) are integrated into a single text plan T_{new}, another text plan T_{out} that overlaps with the integrated plan will remain outside

```
(trailing-comment-01
 ;works for standalone omits or for combined-goal-sequence trees
 (;effects
  ()                                    ;substitution list
  (                                     ;post-code
   (push (make-sequence-tree (list ?t1 ?t2) 'TRAILING-SEQUENCE)
            *CURRENT-TREES*)
   (t-remove ?t1 *CURRENT-TREES*)
   (t-remove ?t2 *CURRENT-TREES*)
   )
  1 ;score function
  )
 (;tree pattern specifications
  (?t1 . (
              ()               ;preconditions about tree
              (?n2 (goal USER ( DO USER ?b)))))
  (?t2 . (
              ((rootp ?n3))             ;standalone tree
              (?n3 (goal USER ( DO USER *b))))))
)
```

Figure 3.1: A rule for adding a text plan to another to create a trailing comment. The $*b$ matches any sub-list of $?b$.

---

the new plan because the scoring function for the applicable rule was too low to allow it to combine. This is typically because an effort to integrate such a text plan would create a message so complex that the heuristic deemed it inappropriate. In the original TraumaTiq output such a message would stand alone; while *RTPI* chooses not to integrate T_{out} into T_{new}, it is still important to acknowledge T_{out}'s relationship to T_{new}. This is because once concepts have been introduced in the integrated text plan, focusing heuristics [McK85b] suggest that other text plans containing these concepts be included in the integrated plan as well. Rather than restructure the result of our transformation T_{new} (against the advice of the heuristic), *RTPI* appends all instances of T_{out} to the end of the message. They are then referred to as *trailing comments*.

Focus [Gro77, McK85a] has been the subject of much discourse research, and it plays several roles in *RTPI*'s generation of messages. Focus theory suggests that when a user's attention is focused on a certain topic, there are a limited number of other topics

Original TraumaTiq messages:
⋆ Caution: check for hemoptysis and persistent wheeze to assess the possibility of a left broncheal injury.
⋆ Caution: check for hemoptysis and persistent wheeze to assess the possibility of a right broncheal injury.
⋆ Caution: check for hemoptysis and stridor to assess the possibility of a tracheal injury.

Integrated *RTPI* message:
Caution: check for hemoptysis and persistent wheeze to assess the possibility of a left or right broncheal injury. Moreover, checking for hemoptysis is also indicated, along with stridor, to assess the possibility of a tracheal injury.

Figure 3.2: An integrated message using the cue word "moreover".

---

which are "easy" for the user to shift his attention to next. These topics include closely related topics and recently mentioned topics. It is the second set, recent topics, which are of interest here. McKeown posits that previously mentioned topics are pushed onto a stack, with the most recent items at the top, and the most coherent focus (topic) shifts will be to items nearest the top. Therefore focus is used to order multiple trailing comments, and they appear in order of the most recently introduced action, thus representing successive pops of a focus stack.

A trailing comment rule, such as the one shown in Figure 3.1, is only loosely specified so that it finds all messages that share topics. In the example shown in Figure 3.2, the first two messages are integrated as described in the previous chapter. The mid-list of the two integrated messages then consists of the actions `(AND Check_Hemoptysis Check_Persistent_Wheeze)`. Thus the third message, which does not share both actions, cannot be integrated into the same structure. However, in the example rule `?b` becomes bound to both actions, and then `Check_Hemoptysis` can bind to `⋆b` (which designates a sublist of `?b`) so that the third message becomes a trailing comment.

Original TraumaTiq messages:
⋆ Caution: check for medication allergies, give antibiotics, set up the auto-transfuser for the left chest tube, insert a left chest tube, and get a post chest tube x-ray immediately to treat the simple left hemothorax.
⋆ Caution: check for medication allergies, give antibiotics, and do a laparotomy immediately to treat the intra-abdominal injury.
⋆ Consider checking for medication allergies and giving antibiotics now to treat a possible GI tract injury.
⋆ Please insert a left chest tube and get a post chest tube x-ray before getting a urinalysis because it has a higher priority.
⋆ Caution: do a laparotomy and repair the left diaphragm immediately to treat the lacerated left diaphragm.

RTPI integrated message:
Caution: check for medication allergies and give antibiotics to treat the simple left hemothorax, the intra-abdominal injury, and a possible GI tract injury. Then set up the auto-transfuser for the left chest tube, insert a left chest tube, and get a post chest tube x-ray to complete treating the simple left hemothorax, and do a laparotomy to complete treating the intra-abdominal injury. Moreover, doing the laparotomy is also indicated, along with repairing the left diaphragm, to treat the lacerated left diaphragm. In addition, insert the left chest tube and get the post chest tube x-ray before getting a urinalysis because they have a higher priority.

Figure 3.3: Five original TraumaTiq critiques and the integrated *RTPI* message. The first three TraumaTiq critiques form a two part intersection (mid and post) and the remaining two are realized as trailing comments (cue words underlined for this example).

3.3 Realization in *RTPI*

The use of a cue word, such as "moreover" in Figures 3.2 and 3.3, can have a significant impact on the coherence of a paragraph. While most of *RTPI*'s improvements to a set of messages are due to the integration of text plans into a new rhetorical structure, the exact wording of the actual text produced from the tree is also important. This will be demonstrated by the examples in the remainder of this chapter. Word selection, both for cue words and reference, is used to help indicate the structure of *RTPI*'s text plan trees

to a reader. *RTPI* also uses other realization techniques to ensure that messages are as unambiguous as possible. In particular, the ordering of message components and the use of definite versus indefinite reference are used to enhance readability, though realization is not a research focus of *RTPI*. These techniques are applied across message types, but there are also techniques that are specific to realizing certain kinds of plan trees (Sections 3.3.2, 3.3.4, and 3.3.5).

A sentence template is a pattern for a sentence that has some open slots for words. Filling the slots correctly will result in a correct sentence, and templates can be efficient when the number of desired sentence patterns is small. Here is an example of a simple sentence template for an omission message, used by both TraumaTiq and *RTPI*:

"Caution: $\#VP$ immediately as part of $\#GER$."

where $\#VP$ and $\#GER$ can be replaced by a verb phrase and a gerund, respectively. *RTPI* currently uses templates for sentence realization for two reasons. First, they are sufficient for this task, and thus the ease of implementation justifies this approach. If *RTPI* is applied in other domains, the need for full syntactic realization can be re-examined in light of the degree of congruency between the domains. Second, the current implementation was designed for real time trauma care, where speed is essential. While templates are an old technology, they are still widely used for these and other reasons [Rei95, Rei99]. Future applications may have different requirements where highly flexible output outweighs the considerations of fast real-time response and rapid development.

3.3.1 Trailing comment realization

The rules that create trailing comments place related text plans in close proximity. However, after this is done, the combined text plans still have repetition of the actions that the trailing comments shared with the rest of the plan. Thus trailing comments that refer to shared actions have the potential to erroneously suggest new instances of actions.

A solution to this problem is implemented in the text realization templates for trailing comments, where the tree structure allows templates to:

1. make the focused action the subject of the sentence, reflecting its *given* status in the discourse;

2. utilize cue words to call attention to its occurrence earlier in the message and to the new information being conveyed; and

3. subordinate other concepts presented with the focused concept by placing them in a phrase introduced by the cue words "along with".

In one such example from the trauma domain (see Figure 3.3), the main text plan contains the communicative goal of getting the user to perform a number of actions, including a laparotomy. A SEQUENCE relation is used to adjoin an overlapping text plan that contains a reference to a laparotomy as a trailing comment, and the reader is encouraged to relate this instance of the laparotomy to the previous one by proximity (the trailing comment has been moved to a position adjacent to the larger message) and by cue words such as "moreover" or "in addition". The cue phrase "is also indicated" is used to remind the reader that the laparotomy has been mentioned before, and does not refer to a new or second instance of an action. That idea is further reinforced with the use of definite reference (further explained in Section 3.3.3).

3.3.2 Scheduling constraint ordering

As noted earlier, trailing comments capture communicative goals that relate to previously mentioned actions, and both trailing comment order and cue words are used to help smoothly shift focus back to the earlier actions. Focus also affects the way in which some communicative goals are realized in messages. For example, if a goal of getting the user to recognize several scheduling constraints is the sole content of a message, it would be realized with the subordinate clause first to call attention to the ordering constraint, as in the following:

> *Before getting the urinalysis, insert the left chest tube and get the post chest tube x-ray because they have a higher priority.*

If this were not done, a long list of preceding actions could obscure the crucial "before" clause. Placing the emphasis on the nature of the message, i.e. that it is about ordering, is accomplished by placing the ordering information first.

However, if the physician has omitted some of the actions and the scheduling constraint is incorporated into the text plan for getting the physician to do the omitted actions, then focus considerations dictate that the main clause appear first since it continues the actions in focus. The following is such an example produced by *RTPI*:

> *Check for medication allergies, give antibiotics, set up the autotransfuser for the left chest tube, insert a left chest tube, and get a post chest tube x-ray to treat the simple left hemothorax. Insert the left chest tube and get the post chest tube x-ray before doing the peritoneal lavage because they have a higher priority.*

3.3.3 Definite and indefinite reference

Messages from the system should be phrased in terms of what is *shared knowledge* in the emergency room. For *RTPI*, shared knowledge is assumed to be the current state of the case, as it has been entered into the computer-based medical record. When a procedure is ordered by the physician, it thus becomes part of this shared knowledge. Consequently, *RTPI* uses definite articles to refer both to procedures and actions already introduced into the treatment plan by one of the system's messages and to entities introduced via the scribe nurse's entry of a procedure or action into the record. For example, even though a peritoneal lavage does not appear in any of the system's earlier messages, a message about a related scheduling precondition will be realized as:

> *Please remember to check for laparotomy scars before you do the peritoneal lavage.*

However, the system may disagree with the physician about whether a procedure is appropriate. Since the use of the definite article suggests an action's acceptance into the

TraumaTiq messages:
⋆ Performing local visual exploration of all abdominal wounds is preferred over doing a peritoneal lavage for ruling out a suspicious abdominal wall injury.

⋆ Please remember to check for laparotomy scars before you do a peritoneal lavage.

Message from *RTPI* integrated plan:
Performing local visual exploration of all abdominal wounds is preferred over doing a peritoneal lavage for ruling out a suspicious abdominal wall injury. However, if you do a peritoneal lavage, then remember to first check for laparotomy scars.

Figure 3.4: An example of a the role of realization in removing the appearance of conflict without violating the social role of the system. This is a repeat of Figure 2.7.

---

treatment plan, indefinite expressions are used when referring to procedures about which there is conflict. For example, if the physician has ordered a peritoneal lavage and the system believes that the need for it is dependent on the results of a chest x-ray, *RTPI* generates the message

> *Do not do a peritoneal lavage until after getting a chest x-ray since the outcome of the latter may affect the need to do the former.*

Articles in trailing comments are also adjusted to reflect the prior introduction of an action. Note that while "insert a left chest tube" is used in the fourth original critique in Figure 3.3 on page 54, the trailing comment is worded "insert *the* left chest tube" to emphasize that this is another reference to a chest tube already mentioned. Similarly, the trailing comment that refers to "laparotomy" uses a definite reference.

3.3.4 Resolving conflict

Section 2.6.4 explains the use of the CONCESSION relation to structurally resolve the appearance of conflict while respecting the system's relationship to the physician,

Caution: check for distended neck veins as part of assessing the possibility of a pericardial tamponade. Next, check for muffled heart sounds to address this goal and also to assess the possibility of a pericardial injury. Then check for continued shock and continued neck vein distention to complete assessing the possibility of a pericardial tamponade.

Figure 3.5: An integrated message with three segments (pre-list, mid-list, and post-list). The mid-list is introduced by "next" in this example.

---

i.e. that of an expert advisor without final responsibility for the outcome of the case. It is important for the realized message to be framed in light of this social role of the system. Consequently, concession relations are realized with the cue word "however" to explicitly convey the CONCESSION relation to the user, i.e. to confirm to the user that the system is aware that what precedes "however" appears somehow inconsistent with what follows. To resolve the apparent inconsistency, the words "if you do $< action >$" are used so that the system's intended relation between the two parts is again explicit. An example of such realization is shown in Figure 3.4. Here the phrase "However, if you do a peritoneal lavage" uses the cue word "however" to explicitly convey that what follows (a peritoneal lavage reference) somehow does not agree with what went before (a local visual exploration reference). The words "if you do" explicitly inform the user that the system recognizes that the user may choose to ignore the system recommendation, at the same time reinforcing the information that what follows (peritoneal lavage) is not the system's first choice of actions. In short, the message is realized to convey that the system recognizes that the final authority and responsibility for care decisions rests with the physician.

3.3.5 Realization of multi-segment plans

The template realization of a three segment intersection (as in Figure 3.5) is governed by the sequences of actions in the segments. If an intersection has two parts (pre-list

Initial integrated message before clause aggregation:

> Caution: check for medication allergies as part of treating the left pulmonary parenchymal injury, treating the compound rib fracture of the left chest, and treating a possible GI tract injury. Then order pulmonary care to complete treating the left pulmonary parenchymal injury, and order pulmonary care to complete treating the compound rib fracture of the left chest.

Result with aggregation in the post-list:

> Caution: check for medication allergies as part of treating the left pulmonary parenchymal injury, treating the compound rib fracture of the left chest, and treating a possible GI tract injury. Then order pulmonary care to complete treating the left pulmonary parenchymal injury and treating the compound rib fracture of the left chest.

Figure 3.6: An example of simple clause aggregation of "order pulmonary care" within an integrated message.

---

and mid-list, or mid-list and post-list) the second part is prefaced with the word "Then". If it has three parts, the second is prefaced with "Next" and the third with "Then" as in Figure 3.5. The use of the cue words "then" and "next" in association with the SEQUENCE relation dates to Hovy's early system [Hov88, Hov93]. While RST does not specify cue words to use when realizing discourse relations, generation systems often use such prototypical associations to simplify the realization process [KD96].

If the pre-list or post-list of an intersection includes parts of two or more messages in it, the system looks for opportunities for aggregation within that part. This kind of aggregation does not require a rule, and so is included here as a realization feature. This is simple clause aggregation, similar to that performed by REVISOR and other systems [CL97, WH96]. Figure 3.6 shows an example of a message where the phrase "order pulmonary care" was repeated unnecessarily. *RTPI* finds the repetition in the plan tree structure and performs aggregation with the result shown. RTPI's search for aggregation possibilities (clause aggregations) within intersections is limited to aggregations that do

A segment that will not be aggregated:

"Do A and B to address X, and do A to address Y"

would become

"Do A to address X and Y, and do B to further address Y"

A segment that will be aggregated:

"Do A to address X and do A to address Y"

would become

"Do A to address X and Y"

Figure 3.7: An example of why aggregation is only performed when the text plan meets certain conditions.

---

not further complicate the text, since some structures are very awkward to realize. For example, within a three segment intersection (one that has pre, mid *and* post segments), clause aggregations are allowed only if the clause aggregations affect all items in that segment. To demonstrate the reason, an example of aggregation that *RTPI* does not perform is shown in the top of Figure 3.7. Here one segment of an intersection contains a possible aggregation, since A has goals X and Y. However, the aggregation would not affect B. The result of this aggregation would reduce the repetition of actions, but would also require that one of the objectives (Y) be mentioned twice, and so it is approaching the form of an action recipe, organized around actions, instead of staying organized around the objectives X and Y. *RTPI* rules emphasize organization by objectives, which is appropriate in the trauma decision-support environment [Cla98], and so this aggregation would not be performed. On the other hand, when the aggregation can affect all of the actions in the segment, as shown in the bottom of Figure 3.7, then text can be aggregated without

Original TraumaTiq messages:
Caution: pad and position the left leg as appropriate and get a neurosurgical consultation immediately to treat the neuropathy to the left leg.

Caution: check for medication allergies, give antibiotics, immobilize the patient, and get a neurosurgical consultation immediately to treat the compound fracture of the vertebra.

Integrated *RTPI* message:
Pad and position the left leg as appropriate as part of treating the neuropathy to the left leg and check for medication allergies, give antibiotics, and immobilize the patient as part of treating the compound fracture of the vertebra. Then get a neurosurgical consultation to complete *both of these goals*.

Figure 3.8: An example with a reference phrase (italicized for this example) that prevents the restatement of two treatment goals.

having to restate an objective.

When one or more actions can be associated with multiple goals, it may be possible to replace the names of the goals with a reference. While the reference phrase in Figure 3.8 prevents having to restate the complicated names of the treatment goals "treating the neuropathy to the left leg" and "treating the compound fracture of the vertebra", the length of text that follows the mention of the first goal is substantial, and may push the first goal well down the focus stack, i.e. it may be hard to remember what the first goal was when the user gets to "both of these goals". This example is a borderline case – it reduces the total number of times that "neurosurgical consultation" is mentioned by one, but it could be argued that the new text is no easier to read and assimilate than the original. Such cases will have to be tested *in situ* to determine their value, since small changes in presentation could favor one text plan or realization over another.

3.4 Evaluation of *RTPI*

While multiple evaluations have confirmed the correctness and effectiveness of various parts of TraumAid, the system has never been actually deployed in an emergency

room, due largely to issues related to the requirement for rapid and correct entry of a large quantity of information about the case at hand into a computer-based medical record [WCC+98]. Thus the integrated messages of *RTPI* have also never been tried in a real trauma setting. While the final evaluation of *RTPI*'s effectiveness can only be measured by deploying it in a trauma bay and evaluating the degree to which its messages change physician behavior, an evaluation of its output can be used to determine its potential benefits and limitations and to identify where further work is needed.

For the evaluation of *RTPI*'s messages, I used actual data originally collected for the evaluation of TraumAid's medical decision-making process. This data was based on 48 collected cases of actual trauma care under a scenario in which messages were produced after each physician order. This scenario arises from a hypothetical implementation where TraumaTiq's messages are being displayed on a large screen in the trauma bay, with the messages changing each time the scribe nurse enters new relevant data. Thus for each case there were multiple sets of messages produced, one set of TraumAid's messages produced for each separate physician order. I extracted one set of messages from the middle of each of the 48 cases and used them in my analysis. [1]

Each message set was used as input to *RTPI*, and messages resulting from a template-based realization of *RTPI*'s text plans were analyzed for conciseness and coherence. There was an 18 percent reduction in the average number of individual text plans in the 48 sets examined. The results for individual sets ranged from no integration in cases where all of the text plans were independent of one another, to a 60 percent reduction in sets that were heavily inter-related. More concise messages also resulted from a 12 percent reduction in the number of references to the diagnostic and therapeutic actions and objectives that are the subject of the trauma domain. The new text plans also allowed

[1] I used a set of messages from the middle of each case since there is nothing to critique at the very beginning of a case and little to critique at the end. It is generally the middle of a case where messages appear in sufficient number to consider the effect of *RTPI*.

some treatment goal names to be replaced by references during realization, making the messages shorter and more natural in appearance. This numeric analysis confirms my hypothesis that a rule-based system can effectively make a set of messages more concise and less repetitive.

While the numeric analysis shows that the rules are clearly effective at accomplishing my intentions with regards to conciseness and coherence, this numeric analysis is not sufficient to show that the rules would produce text preferred by human readers. In other words, a coherence rule may be applied to improve coherence in some theoretical model without improving coherence from the perspective of a reader. Thus it was also necessary to present the text produced by the system to human readers.

To evaluate coherence, results from twelve cases were presented to three human subjects not affiliated with our project. The evaluation examples consisted of the first eleven instances from the test set where *RTPI* produced new text plans, plus the first example of conflict that appeared in the test set. Results were presented as randomly ordered blind pairs of *RTPI*'s message and TraumaTiq's message set that was used as input to *RTPI*. The written instructions given to the subjects instructed them to note whether one set of messages was more comprehensible, and if so, to note why. Two subjects preferred the new messages in 11 of 12 cases, and one subject preferred them in all cases. All subjects *strongly* preferred the messages produced from the integrated text plan 69% of the time. In general, the larger the original message set, the more the subjects preferred the integrated message.

The two subjects who each preferred the original message set for a single case did not choose the same case. In one case, the original and integrated messages each made reference to both "rule out a pericardial tamponade" **and** "rule out a pericardial tamponade urgently". The appearance of both of these treatment goals in one set is an artifact of the TraumAid system's representation of treatment goals. In the original messages, the subject simply crossed out the similar message and noted "repeat" next to it. In the

Original TraumaTiq messages:
Caution: check for distended neck veins, muffled heart sounds, continued shock, and continued neck vein distension to assess the possibility of a pericardial tamponade.

Consider checking for muffled heart sounds to assess the possibility of a pericardial injury.

Integrated *RTPI* message:
Caution: check for distended neck veins as part of assessing the possibility of a pericardial tamponade. Next, check for muffled heart sounds to address this goal and also to assess the possibility of a pericardial injury. Then check for continued shock and continued neck vein distension to complete assessing the possibility of a pericardial tamponade.

Figure 3.9: An example from the evaluation in which the *RTPI* message was not preferred by a subject.

integrated set, however, the text put the two treatment goals in the same sentence, and the combination "rule out a pericardial tamponade and rule out a pericardial tamponade urgently" was noted, and marked by the user as "bad". A later version of *RTPI* was able to detect such problems and correct them.

Figure 3.9 shows the second case where the original message set was preferred by one subject. In this example two relatively short original messages are presented together, and the integrated message was marked by the subject as "muddled". Because the original message set is straightforward and not long, the complexity of the three part message is not seen as beneficial. From a medical perspective, this message is suboptimal because a physician making a critique would not refer separately to a pericardial tamponade and a pericardial injury[Cla98]. The fact that both appear here is an artifact of TraumAid's representation of treatment goals. Of the two subjects who preferred the integrated message, one noted the original set's repetition of "muffled heart sounds" and said that the two "pericardial tamponade" references should be related, and marked the integrated message

"better"; the other subject noted the repetition in the original messages, and marked the integrated message as "better, though not terribly much" since the original was "fairly good".

This human evaluation, while small in scale, shows clearly that the texts produced by *RTPI* are preferred by human readers. This result confirms the function of the rules in *RTPI* by showing that when the rules are applied to remove conflict, improve conciseness, and enhance coherence, the resulting texts are preferable to the original message sets.

3.5 Summary

The last class of text plan integration rules are those that re-order messages to place related messages in close proximity. This enables the system to perform realization in a manner that facilitates correct interpretation of the message set as a whole.

While realization is not a research focus of *RTPI*, correct realization of integrated text plans is essential to realizing the potential of *RTPI*'s plans. Re-ordering to facilitate focus shifts, and the choice of definite/indefinite reference are both used to reduce ambiguity and emphasize the intent of the plans. Other techniques are designed to improve the realization of particular kinds of text plan trees, especially the use of cue words and phrases to convey text plan structure.

Evaluations to date[2] indicate that *RTPI*'s integrated messages are preferred by human readers and a significant improvement over the original messages. These included a numeric analysis of *RTPI*'s messages compared to the original message sets from TraumaTiq, as well as an evaluation by human readers to confirm that the application of *RTPI*'s rules did in fact produce output preferable to the original messages.

Thus I conclude that the implemented system described in Chapters 2 and 3 confirms my hypotheses described at Chapter 2's beginning. In particular, *RTPI*:

[2] A full evaluation must wait for deployment of TraumAid in a live trauma bay, which is not pending.

- made a set of text plans more concise by reducing repetition, thus producing messages that were preferred by test subjects;
- took sets of text plans that appeared to be in conflict and re-organized the text plans by exploiting relations between them so that test subjects agreed the new messages are preferred, while respecting the social role of the system; and
- demonstrated that these improvements were computationally feasible in a rule-based system.

RTPI is an obvious precursor to the system introduced in Chapter 4. MADSUM operates in a domain where there are no sets of actions associated with goals, but a single go/no-go action choice. While very few of RTPI's rules are applicable in such a domain, the algorithm and the domain independent text plan integration show MADSUM's roots in RTPI.

Chapter 4

ADAPTIVE RESPONSE GENERATION FOR DECISION SUPPORT IN A DYNAMIC ENVIRONMENT

The previous chapters focused on the problem of generating effective messages in real-time decision support. I presented my solution, RTPI, a text generation system that takes into account an arbitrary and often inter-related set of communicative goals and produces a message that realizes the entire set in a concise and coherent manner. RTPI takes into account the purpose of the messages, the situation in which the messages will be received, and the social role of the system.

However, these abilities are not sufficient for some domains. Consider a decision support system that provides texts about financial choices, such as stock purchases, to agents of an investment advisement firm, the ABC Corporation. ABC's financial advisors use the generated texts as information for themselves and for clients. They want to be able to request a report generated at low cost, using generally available information, to be included in a letter mailed tomorrow, or a high cost report using guaranteed-accurate, up-to-the-minute data to be delivered in a meeting that starts in ten minutes. All reports must be tailored to the specific information needs of the application at hand. Having requested a report, an advisor may wish to modify any of the specifications of the request without having to restart the information gathering process. The sources of the information may be internal or external agents (websites, human experts, financial reports), may be distributed across a wide communication network, and can provide a variety of responses to an information request that vary in topic, length, quality, cost, and time to delivery.

Sources that are contacted for information may fail to respond, may promise to deliver information and then fail to do so, or may produce information that fails to meet the original specifications. Furthermore, for reasons of information privacy, computational efficiency, and system redundancy, computation of parts of the integrated result message is distributed.

This task requires the integration of information from a variety of sources, as in *RTPI*, but now the information available from the decision support environment is more dynamic and more varied. While *RTPI* provided support regarding the performance of a fixed set of possible medical procedures within a specified protocol for an prototypical user with known preferences (the expert physician), this task requires the representation of both changing user preferences about the content and attributes of the final message, and a means of representing the trade-offs required when user requirements are in conflict. For example, the decision support system must be able to represent and manage a user's preferences for low cost, high quality, and immediate delivery when no single result can provide all three. Given such a representation of user preferences and message attributes, operation in a dynamic information environment requires a dynamic system that can facilitate the construction of a response that uses such a representation, while flexibly responding to the changing environment.

To be successful in this domain, a decision support system must be adaptable to a wide variety of users in different situations. The kinds of information that will be most useful for decision-making will vary depending on the individual. For example, in the financial domain, some users might be most concerned about the riskiness of an investment while others are more interested in an investment's prospects for financial gain. In addition, constraints or priorities on resource usage will differ. For example, some users would be willing to pay substantially for an expert opinion, while others would prefer to spend less money even though their information will come from a generic news source. Different individuals will have different priorities, and those priorities will change

over time.

To understand why user priorities are essential to decision support, consider a system that does not take them into account. For example, a system that does not vary its responses to reflect user constraints on length may provide the user with a message that the user does not have time to read, memory to store, or room to display for viewing. A system that ignores the monetary cost of a response might arrive at a message that the user cannot afford (or, conversely, a message so cheap that the user will not value it). And a system that does not consider time may provide a message whose usefulness expired before its delivery (e.g. a message about a stock purchase that gets delivered after the close of the market).

The situation is complicated when the decision support environment is not static. A user's status may change, or the environment could change in a way that affects a user's priorities. Suppose, for example, that a user sends a request for decision support from a desktop computer with a large screen, indicating a preference that the resulting message not require more than one page (about 300 words) to display. Just after sending the request, the user's schedule changes and he learns that he will be on the road and will have to read the response on his Blackberry™. In this way environmental change (the user's physical location, and thus the size of the display device available) has affected the user's priorities: the user now strongly desires a more concise message.

Similarly, the environment may change in such a way that it is difficult to predict the outcome of the information gathering results that are central to performing decision support. For example, in the financial domain, an online financial report may rely on a daily survey of stock brokers. However, if there is sudden market volatility (e.g. a crash), the brokers may be too busy to answer the daily survey, and the information gathering task fails (alternatively, the price of that information could rise substantially, making the outcome change in a different way).

In addition to information outcomes that are difficult to predict, the resources consumed by information gathering tasks can also be difficult to predict. Consider space as a resource, e.g. the number of words that will fit on a screen. In the financial domain, a report that usually allocates approximately 40 words for a sentence about a stable stock might have to change that allocation if the stock is suddenly the focus of a corporate acquisition battle.

Thus the problem is to

1. develop a response generation methodology that takes into account information about the user, the user's resource constraints, and the user's priorities (with regard to information content and resource usage); and

2. make the methodology responsive to internal (user) and external (environmental) dynamic factors.

My hypothesis is that a computational methodology can be developed to generate and integrate communicative plans for decision support in an environment that has the following dynamic characteristics:

- user priorities can change over time;

- it is difficult to predict the resources consumed by an information gathering action; and

- it is difficult to predict the attributes of information gathering results.

Furthermore, such a methodology should be designed to perform quickly in the presence of such dynamism so that computational decision support in such an environment is shown to be a realistic goal.

More specifically, I hypothesize that the user information, constraints, and priorities can be captured in a user model that includes a decision-theoretic utility function. The function can then be applied to evaluate candidate decision support messages under

conditions of changing user information. Furthermore, I hypothesize that a distributed, multi-agent approach will provide the flexibility and responsiveness required to generate support under these dynamic conditions. The remainder of this chapter outlines my approach: a decision-theoretic agent-based approach to response generation for decision support that ranks different possible full responses according to their value to the user, and takes into account both resource and content attributes in doing so. The system assumes a dynamic environment and weak predictive models of ultimate information utility, and thus requires dynamic organizational management in response to run-time results, necessitating a distributed, adaptive system. This methodology has been implemented in the system MADSUM (Multi-Agent Decision Support via User Modeling).

4.1 A Brief Overview of MADSUM

MADSUM is the result of my investigations into 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. MADSUM is currently implemented in the financial investment domain, where it produces messages that support a user considering a specific investment. The system design combines approaches from both user-modeling and agent architecture. The implemented system consists of a hierarchy of cooperative agents that use a negotiation process to solicit and organize other agents to produce information, and a presentation assembly process to coherently assemble the information into text for decision support. At each stage the decisions of the agent consider the preferences and constraints represented in the user model.

4.1.1 Multi-Attribute Utility Function

The user model includes a multi-attribute utility function. Attributes are chosen for a particular domain, though many domains share attributes such as dollar cost. Constraints are hard limits on the possible values of attributes; they are set by the user to prevent the system from generating unacceptable results. Constraints are employed to determine

if a particular result is acceptable, but provide no guidance to the system about which acceptable decisions are preferred over others.

In contrast, the utility function allows the agent to weigh the benefit of different decisions about resource usage and information selection that do not violate constraints. In other words, utility provides a means to evaluate decisions about the message, while constraints limit the decision space. The user of the system can tailor the utility function to affect:

- the information content of the result (which kinds of information are included, and how much);
- attributes of the resulting message itself (such as text length and cost); and
- attributes of the planning process (e.g. time).

This 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.).

4.1.2 Agent Architecture

An agent architecture is appropriate for this problem for several reasons. First, the nature of the information gathering problem is distributed. For example, multiple sources of the low-level information required for investment decision-making (e.g. numeric financial information about companies) are freely available on the web, but rarely is all the required information available from a single source. A distributed architecture can solve this class of problems utilizing parallelization, distributed expertise, and fail-soft performance[Jen95a, SV00].

Second, MADSUM is intended to be easily adapted to new domains. The distributed approach is an ideal way to decompose a decision support task so that previously designed MADSUM agents can be re-used without modification in domains that require

expertise which overlaps with previously implemented domains. Also, new agents can be easily added to existing domain implementations as new sources or new areas of content expertise are added (this can facilitate the incorporation of legacy code).

Third, in agents as in business, distributed decision-making is ideal for rapid reaction to a dynamic environment[WP82], since decisions can be made by the part of an organization closest to the issue being considered, without extensive communication and negotiation along hierarchical lines. In the case of decision support, the dynamic aspects of the agent environment can be thought of as consisting of both the external world of information and information sources, and also the internally-modeled user preferences regarding constraints and priorities. The problem is therefore dynamic in both external and internal dimensions, increasing the importance of a distributed decision-making process.

Finally, the use of a hierarchy of independent agents avoids the information bottleneck associated with having all information processed by a single agent. Reducing the size of the problem makes individual decision makers simpler and more reliable[Jen95a] and can facilitate parallelization.

4.2 The Financial Investment Domain

I have applied the MADSUM architecture to decision-support in a financial investment domain. MADSUM provides investment information to a user making a buy/don't-buy decision on a single investment, i.e. a specific amount of a certain instrument at a certain price. The MADSUM decision making algorithms and the agent hierarchy, communications, and interaction are domain-independent (see Chapter 6). Extending MADSUM to a new domain requires three steps. First, the set of attributes in the user utility function must be altered to reflect the kinds of user preferences that are relevant in the new domain. For example, decision-support in a restaurant domain might require utility function attributes regarding decor and food quality[MFLW04].

The second step is the implementation of a set of domain-dependent information agents. 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 current personal and financial status. In MADSUM these are "wrapper" agents that act as interfaces between the actual information source (possibly a website, a database, or an external agent) and the MADSUM system.

The last domain-dependent step is the mapping of the domain information into text plan templates (see 5.5.2.3) so that the wrapper agents can produce text plan trees for the task agents above (and eventually the Presentation Agent) to integrate.

As noted earlier, MADSUM is implemented in a financial investment domain. I chose this domain for the following reasons:

- While full-scale investment advisement involves developing a financial plan, the plan eventually decomposes into binary buy/don't buy decisions that reduce the complexity of the decision support problem.

- Financial domain decision-making typically includes a large numeric component, which is composed of the computable answers to multiple, independent questions about a particular investment or investment strategy. This facilitates the design of agents with financial expertise, since there exists an accepted body of standard methods of numeric analysis. (Contrast this with decision support for buying art, or adopting a child.)

- The financial investment domain has a huge web presence (typing "investment advice" into Google recently retrieved 21 paid listings and over 17 million results), thus providing many examples of actual information sources available and their products.

- Many people are interested in making investment decisions to some degree due to the prevalence of 401K accounts and similar vehicles that encourage market investments.

- It is a domain with which I am familiar, due to my experiences in the banking industry and contacts with people in the field.

The financial investment domain information can be viewed from many perspectives, resulting in a wide variety of classifications. For the purpose of implementing MADSUM, I chose three broad content areas for which information would be generated: investment risk (hereinafter RISK), potential for increased value (VALUE), and the impact of the investment on the user's financial goals (GOAL) as stated or hypothesized from the user model. Within those three areas, I selected standard financial analysis measures (all available from free financial websites) to be the information provided by MADSUM source agents.

4.3 The User Interface

A user employs MADSUM via a graphical user interface, and the system responds with a text message in English (see Figure 4.1). To minimize the complexity that the user sees at any given point in the interaction, the interface has three distinct parts, each a separate screen. The first screen is for entering user data (e.g. portfolio allocation goals) that will become part of the user model. The second screen allows the user to enter basic preferences and constraints, and then send a request for decision-support to the system. The third screen is a more detailed interaction screen for advanced users. Once the first screen has been used to enter initial data, most user interactions will involve only the second screen.

The "Basic" screen of the interface is the heart of the user-MADSUM interaction. Figure 4.2 shows the graphical sliders employed by the user to indicate relative priorities

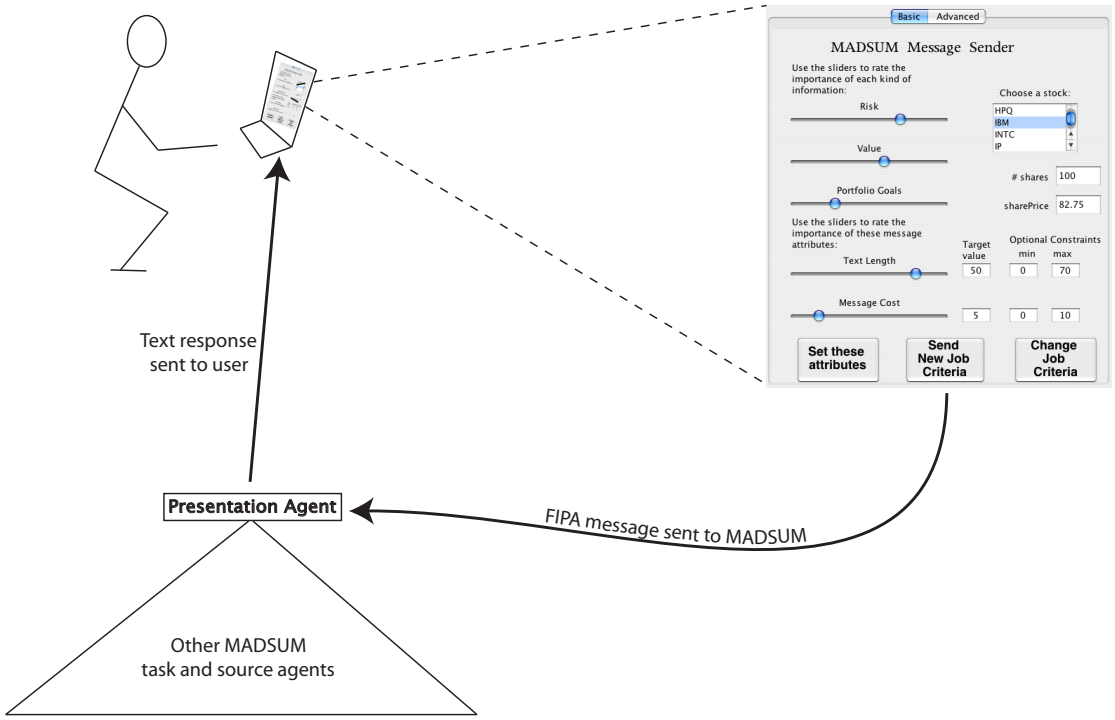


Figure 4.1: Using *MADSUM*– the user enters information, preferences, and constraints on a screen, which are sent to the Presentation Agent (PA). The PA returns the decision support message to the user's screen.

for the attributes in the utility function (these are similar to the meta-sliders used by Wagner et al [WGL97]). The user can also enter fixed numeric high and low constraints for the attributes, though these come with preset defaults.

This screen also presents the user with windows for proposing a particular investment for which they desire decision support. The user enters a stock symbol, a number of shares, and a price per share. Clicking the "Send" key forms a message[1] which is sent to the MADSUM Presentation Agent to begin the process. At the conclusion of the process a new screen shows the resulting text message from MADSUM.

More advanced features of the MADSUM interface are accessed via the third

[1] see Chapter 6 for information about the messages used by DECAF agents.

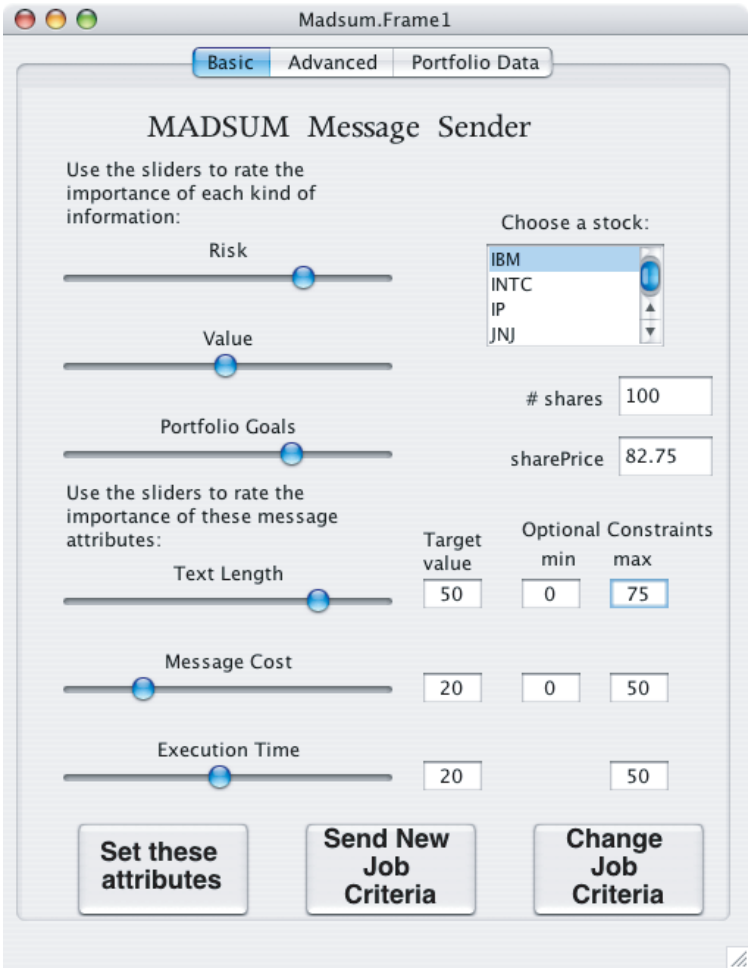


Figure 4.2: The basic MADSUM interface for user input.

screen (Figure 4.3). Here the user can modify the separate functions that control how utility is derived from each attribute term (see section 5.4.4). This allows utility for different attributes to have soft constraints or various other behaviors over the range of the attribute.

4.4 Summary

Individuals differ not only in the resources they have available to expend on information, but also in the priorities they place on different kinds of information. My hypothesis is that effective decision support can be provided by representing these differing

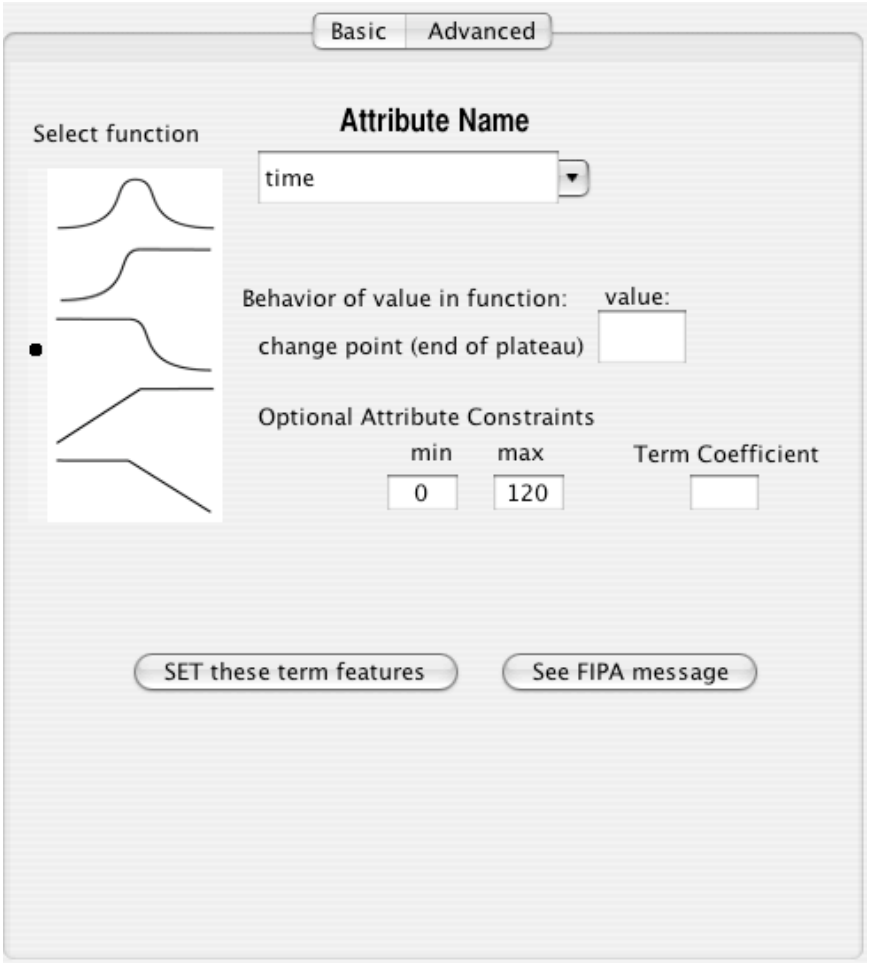


Figure 4.3: The advanced MADSUM interface for user input.

priorities and related constraints in a user model, and then using that model to allocate resources for an unseen task across multiple agents in a dynamic environment. MADSUM is my implementation of this proposed methodology; it is a distributed adaptive system that uses a negotiation process to solicit and organize agent responses to produce information, and a presentation assembly process to coherently assemble the information into text for decision support. Chapter 5 explains how a user model, including preferences and constraints on both content and the resulting message, informs both processes. An evaluation demonstrates that the influence of the user model on content selection and presentation improves system output. Chapter 6 then describes the aspects of the agent

architecture that allow MADSUM to dynamically adapt to the runtime environment, and includes experimental data showing that the architecture responds appropriately and predictably in the presence of inevitable information failures.

Chapter 5

A DECISION-THEORETIC APPROACH TO RESPONSE GENERATION

5.1 Introduction

As discussed in the previous chapter, an effective decision-support system should produce responses that are tailored to the individual user. In producing such responses, the system must not only take into account a user's preferences for different kinds of information but also a user's priorities and constraints on the usage of different resources. This chapter presents my solution to adaptive response generation, which takes the form of a decision-theoretic approach that utilizes a formal utility function to rank different possible full responses according to their value to the user and that takes into account both resource and content attributes.

I first discuss work in areas related to the generation of responses that are adapted to a user's preferences. I then explain how MADSUM models user preferences regarding types of information content included in a response and attributes and constraints of the response itself. Finally I present evaluations of two aspects of MADSUM's output, showing that subjects presented with a user's preferences agreed with MADSUM's decisions regarding both content selection and ordering.

5.2 Related Work

5.2.1 Early Work on Adaptive Response Generation

MADSUM follows work in other systems that adapt their responses to an individual user. The ADVISOR system [MWM85] operated in a domain of student advisement,

giving information about what courses a student should or could take to satisfy a particular goal. The system inferred a student's goal from a discourse segment, ranking the goals as either *definite*, *likely*, or *plausible* based on whether they were explicitly noted or could be inferred. One of several information hierarchies was selected based on the detected topic of the last question and the current *relevant* user goal, and the different contents of the hierarchies triggered different rules, which determined the output.

Paris [Par88] modeled a user's level of expertise, and varied a response not only by modifying the content, but also by choosing a process-based or object-based description. MADSUM does not make any high-level decisions about how to present information, but leaves that determination to agents that create a text plan about a specific piece of information at a low level. In principle, at least, style choices could be made at that level in a manner similar to Paris'.

McCoy [McC88] used a detailed, pre-existing user model to design a response when the system detects that the user has a misconception. This work addressed the "overlay assumption" issue, i.e. it considered the possibility that the user's knowledge/beliefs are not a subset of the system's knowledge, but may simply intersect.

All of these early systems modified the system's response based on a user model, either pre-existing or acquired. Section 5.2.2 considers work that specifically models the preferences of a user (as opposed to e.g. a user's goals, knowledge, beliefs, or attention state) and work that uses those preferences affect results in a decision theoretic manner.

5.2.2 Decision Theoretic Response Generation

Decision theory posits that in complex systems, any outcome preferred by a user is preferred for more than one reason. In a "decision theoretic" system the *utility* of a good or service can be calculated by considering the attributes of a good or service in light of a mathematical representation of a collection of preferences. Formal utility functions capture this idea in an equation with multiple terms, where each term represents one attribute about which a user has preferences. Adaptive systems have used concepts of

utility theory, either informally or formally, to make decisions that take into account the user's preferences.

In Section 5.2.2.1 I note two systems that used models of user preferences to affect the system's communications to the user. More recently, researchers have begun to design their responses to incorporate user preferences by using formal multi-attribute utility functions. Section 5.2.2.2 reports on work in this area.

5.2.2.1 Modeling User Preferences

A very early system using a model of user preferences, GRUNDY [Ric79a, Ric79b] employed stereotypes to instantiate individual user models to make book recommendations. A design goal of GRUNDY was to be able to make a recommendation without asking an exhaustive list of questions. Individual facts, such as gender, would map an individual reader to a stereotype to imply preferences for or against books that were romantic, violent, intellectual, etc. The system could then make assumptions about whether a reader would enjoy a book based on the preferences in the stereotype. By using the stereotyped preferences, the system could make a recommendation after asking fewer questions than would be required if the system did not employ a stereotype.

Elzer et al [ECCC94] focused on identifying user preferences about university courses dynamically during a dialogue. Her system could identify preferences stated by a user, or infer them by examining a selection of candidate options rejected by a user. Once the system was confident that it had a sufficient model of the user's preferences, it was capable of suggesting an action (taking a particular course) that better fit those preferences than the user's proposed action.

5.2.2.2 Tailored Response Generation

Utility functions offer a simple means to apply information about a user's preferences to a situation. Formal utility functions and Multi-Attribute Utility Theory (MAUT)

date to the 18th century[Bal08] and are commonly used in game and decision theory. MADSUM uses a typical form of utility function

$$Utility = \sum_{i=1}^{n} weight_i function_i(attribute_{value_i}) \tag{5.1}$$

in which different attributes (such as size or cost) of a situation or problem are each given weights that characterize their worth (or utility) to an entity. The function in each term usually maps the attribute value into some predefined numeric range, such as zero to one.

Chu-Carroll [CCC94] has dialogue agents collaborating in a recursive *Propose–Evaluate–Modify* cycle to form a plan. Chu-Carroll did not use the term "utility", but in fact her method for evaluating plan alternatives is a summation of attribute preferences multiplied by a value (a distance from a target), or what is now called a utility function. Lesh[LHL97] uses a similar model in ranking candidate flights in a travel domain.

One difficulty associated with using utility theory in practice is determining the user's preferences, i.e. the weights and functions to associate with each term in the utility function. *MADSUM* facilitates user input by employing graphical sliders, but if a system had a very large number of utility function terms this would become unwieldy. GRUNDY [Ric79a] reduced the number of questions a system needed to ask before performing, but recent work in this area has focused on reducing information acquisition or requirements in provably optimal ways.

One recent effort at identifying and ranking user preferences re-examined the problem of asking the user questions to determine preferences. The Iona system [CP01] uses utility theory to make travel choices for a user. The system starts by finding the user's absolute preferences, i.e. opinions held by the user which are not subject to change (e.g. "a window seat is always better than an aisle seat."). In each consecutive step in the decision process, the system chooses the next question based on a calculation of maximum information gain (in this case, the best reduction in utility uncertainty). The questions about which the user feels most strongly will distinguish among choices more rapidly and clearly than questions for which the user does not have immediate or strong preferences.

Thus the Iona system uses utility in two ways (neither of which is the way *MADSUM* uses utility): to evaluate a possible travel choice, and to evaluate the "usefulness" of the system's next query to the user.

The MAUT Machine [SDB03] (for **M**ulti-**A**ttribute **U**tility **T**heory) is not a complete decision support architecture, but rather is designed as a component that provides utility-related services to a "recommender system" that helps a user make choices from a product catalog. The first service is an expert interface that allows the implementation designer to design the utility function and the individual functions that compute the value of each term, and then to "mark up" the products in the catalog with the information required to calculate an item's utility to the user. The second service is the application of the Analytic Hierarchy Process [Saa80] which organizes utility terms hierarchically, enabling the system to minimize the number of questions required to elicit user preferences. Finally, the MAUT machine calculates the utility of a product to a single user, or to a group of users. Again, this design as currently implemented does not calculate the utility of the information provided to the user, but rather the utility of the product being considered for purchase.

These two systems implement technologies that could offer a different way for users to express preferences. This could be valuable if *MADSUM* were required to use a vocal or keyboard interface instead of a graphical one. Integration of work of this kind could conceivably enable the elicitation of more preferences, resulting in a more complete representation of user utility.

A slightly different approach to determining the weights in the utility function is used by MATCH [WWS$^+$04, JBV$^+$, SWWM02]. In this work, instead of directly setting numeric weights for each term in the function, a user's preferences are represented in a tree with attributes as leaves. The term weight for an attribute is then the product of the weighted arcs on the path from the root to the relevant attribute leaf[CM00]. Once

the weight is calculated from the tree the function has the standard form shown in Equation 5.1.

MATCH consists of a sophisticated user interface on specialized hardware for restaurant choice based on user preferences captured in a utility function. The architecture is intended for use in other domains as well; however, several design choices limit the kinds of domains for which MATCH would be suitable. In particular:

1. Mapping of attribute values is uni-directional, e.g. increased cost is always bad; while this makes some sense in the restaurant domain, it is not suitable for attributes in other domains. In contrast, MADSUM allows a user to express that increased length is good up to a point, then bad beyond that point.

2. MATCH employs the SMARTER [EB94] procedure for eliciting multi-attribute decision models, which uses ordinal assignment of weights. Since weights must be ordered, a user cannot express e.g. that risk and value information are both of equal high value (because two equal weights are not ordered with respect to one another), or that risk information is *highly* valued relative to value information (since ordinal data does not include magnitude). Both of these relationships can be expressed in MADSUM. While this is not a critical issue in a domain like restaurant choice, it could be unsatisfactory to a user making critical decisions about money or health options. The SMARTER procedure has the advantage that it results in the user's preferences being coded in a "value" tree with weighted arcs, and weight of an internal node can be used to represent the user's attitude about the children of that node. However, this advantage also results in the need to correctly design the original hierarchy so that internal nodes are meaningful. For example, in the restaurant domain, MATCH originally grouped decor, neighborhood, and service together under a node called "ambience"; however, this grouping was not considered intuitive by early users.

3. The MATCH utility function computes the utility of each decision outcome (i.e. the utility of a particular choice) to the user, *not* the utility of the system's *message* to the user. MATCH selects the highest utility choices (e.g. restaurants) for discussion, limiting the number of propositions expressed based on a user-set conciseness constant (k). Once the choices that will be discussed are determined, the propositions about each choice are then selected based on how *compelling*[KS94] they are. An attribute value's compellingness is based on the difference between the utility, to the user, of the actual attribute value and the mean attribute value for all choices in the restaurant population being considered. If an attribute value's compellingness exceeds $k\sigma + compellingness_{mean}$, where σ is the standard deviation of the compellingness of the population, then the proposition about that attribute will be expressed. Note that choices with poor utility will have low compellingness, so that if k is sufficiently high that some propositions are excluded, it will be propositions about choices with low utility. Thus the system cannot reliably deliver messages that warn of particularly poor choices, even if such a message would be of high value to the user, since such propositions may be excluded from all but the most verbose responses.

4. MATCH "agents" are not independent decision makers, and thus not autonomous agents in the sense of [WJK99]. Rather, they appear to be separate program functions that can run on different parts of a network if necessary, passing information via XML structures to achieve a traditional pipeline architecture. For example, the text planning module can run on a different machine than other modules, making the system distributable, but it does not perform its task in parallel with other modules, even if they are running on separate machines.

MATCH is principally an argumentative system, making a choice and attempting to convince the user that the choice is correct. Since persuading, not informing, the user is the primary goal, the system always *argues* (says something evaluative) about values

that are supportive of the system's choice, but only *expresses* a value when mentioning arguments that do not support the system's recommendation [CM00]; thus the user could be deprived of information that could help the user make an informed choice. This would not be acceptable in a system where the user bears responsibility for critical outcomes, as in *RTPI* or *MADSUM*, but works well in the restaurant selection domain.

To enhance effectiveness, early versions of MATCH [CM01, CM99] used content ordering strategies suggested by argument theorists such as [MG96]. However, these strategies are not quantitatively derived or experimentally validated, but are descriptions of strategies the authors believe to be effective. Later versions [WWS$^+$04] arranged paragraphs comparing restaurant choices by presenting them in order of utility, and the details about each choice were presented with the choice, and ordered to maximize succinctness after aggregation.

MATCH was tested in two separate domains, real estate (house selection) and restaurant choice. While the content ordering strategies of the systems varied slightly, the content selection strategies were the same, and both systems showed that the output tailored to user preferences was better than non-tailored output. The first system's evaluation showed that tailored output affected user attitudes, while the second system's evaluation showed that subjects preferred the tailored output when viewing a hypothetical dialogue where the system's responses were tailored to the subject's preferences.

FLIGHTS ([MFLW04]) uses a multi-attribute utility function in a user-model to influence the spoken language generation of airline flight recommendations. It first chooses flights of interest to a user by utility order, but does not necessarily include all of these high utility choices in its message. Once the highest utility choice is mentioned, other high utility choices will only be mentioned if they have a *compelling*[KS94] attribute that has a value different from that of the highest utility choice. Thus if the system considers a flight very similar to a previously mentioned option, it will not be mentioned,

even if the utility of that flight to the user is higher than another flight that will be mentioned. For example, the high utility choices in order might be flights A, B, and C; but if B has no compelling attributes that are different from A's, B will not be mentioned. While this strategy is appropriate for the flight choice domain, it may be inappropriate for a domain where the user was ultimately responsible for a critical outcome, such as medical care decision support, and would like to be made aware of all options.

After flights are chosen to appear in the message, a content planner selects schemata which then determine how the attribute information will be presented. The output is in the form of RST-style trees, which are passed to a domain-specific sentence planner that uses templates. The output of the sentence planner is realized for speech and text using a Combinatory Categorial Grammar (CCG)[Ste00].

Like MADSUM, FLIGHTS applies the influence of the user-model at multiple stages of the process. But in FLIGHTS, as in MATCH, the user-specific weights are strictly for domain information, such as whether a business class seat is preferred. In contrast, *MADSUM* allows the user to establish preferences about the attributes of the message itself.

The utility functions used by MATCH and FLIGHTS both include functions in each term of the utility function that map the values of each attribute to a $[0..1]$ space. Unlike *MADSUM*, however, the individual term functions are not selected by the user, but instead are chosen by the program designer.

While each of these systems makes use of a utility function, they each use the 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 allows the user to influence attributes of the system's output, in terms of cost, length, and time, as well as domain-specific preferences such as topic. Thus MADSUM is concerned with the total utility of the *message* to the user, including the utility of the information provided as well as the the utility of the attributes of the presentation.

Placing the focus on the message allows MADSUM to represent that the utility of the message is not the same as the desirability of some result. For example, if a user proposes a very poor investment choice, MADSUM can represent that a message advising against that choice has very high utility, even though the utility of the investment to the user might be low. Conversely, telling a user that an excellent investment *is* an excellent investment may have low utility, if the information is not novel to the user. Of course, there are many other circumstances which could be taken into account (the degree of confidence of the user, the degree to which the user expects novel information, etc.) but these examples are sufficient to show that decision support message utility is distinct from the utility of the result of a decision.

5.3 Modeling User Preferences and Priorities in *MADSUM*

A user affects MADSUM's message output by specifying preferences and priorities with respect to a set of predetermined message attributes. This specification is performed in the user interface (see Section 4.3). MADSUM is designed to utilize an arbitrary number of attribute preferences, since the number of attributes will vary for different domains. For my implementation I selected the attributes of text LENGTH, dollar COST, and execution TIME, and three domain specific attributes, one for each of three topics in the financial investment domain (see 4.2). These topics are RISK (the riskiness of an investment), VALUE (the prospects for the investment gaining in value), and GOAL (how the investment relates to the individual's portfolio allocation goals). Agents categorize all information as belonging to at least one of these topics, and lower level agents are grouped by topic area (see 5.6).

All information about user preferences from the user interface becomes part of the user model. The next section explains the user model and its components, and the ways in which these components affect MADSUM's message output.

5.3.1 The User Model

Adaptive response generation requires a model of the individual user. User models can have both long-term and short-term components [KF88]. Long-term components are assumed constant during a single decision support task, and include data such as user age and portfolio allocation goals. Short-term components may vary during the user-MADSUM interaction, and include preferences like length and cost.

MADSUM was designed to exploit a user model in the context of multi-agent decision support. The user model has three components: User Attributes (long-term), Constraints (short-term), and a Utility Function (short-term). User Attributes are captured in a long-term user model that is constructed once (but could be revised over time), while the Constraints and the Utility Function will vary with different user interactions and perhaps even change during an interaction. All three of these are initially constructed, and any modifications made, via the user interface (see Section 4.3).

5.3.2 User Attributes

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
- existing investment portfolio
- portfolio allocation goals (Portfolio allocation goals refer to an individual's desired distribution of investment categories, such as stocks, bonds, or cash equivalents.)

If not expressly given, portfolio allocation goals can be hypothesized based on a simple stereotype ([Ric79a]) derived from an individual's personal characteristics such as age, salary, and years to retirement. For example, investment advisors typically recommend that people close to retirement age maintain a progressively smaller percentage of their portfolios in stocks.

The User Attributes are used by agents to estimate the significance of certain pieces of information (see 5.4.4). For example, if a proposed investment would cause one's investment portfolio to deviate from one's portfolio allocation goals, information about the deviation becomes more significant as the deviation grows.

5.3.3 Constraints

The Constraints component of the user model offers the user the option of setting hard constraints for a given attribute. Hard constraints are values that an attribute *must not* exceed in a response, and are used to pare the search space before utility is calculated. For example, if the user sets 75 words as the *hard constraint* for length of the response (perhaps because he is using a handheld device with a miniature viewing facility), then longer responses will not be considered by the system.

The user also has the option to set soft constraints for certain attributes. Soft constraints are attribute values that the user would prefer not be exceeded in constructing a response. If the user sets 75 words as a *soft constraint* rather than as a hard constraint, then the estimated utility of the response will depend in part on how much the length of the proposed response exceeds the soft constraint. Thus soft constraints are a kind of preference on resource use, and exceeding them decreases utility (as in "I'd rather the answer didn't exceed 75 words, but if all of the information is really important it's ok"), whereas hard constraints prevent an option from being considered (as in "Under no circumstances will I wish to see a message longer than 75 words"). Soft constraints must be implemented as part of the utility function to operate in this manner, and are further discussed in 5.4.3.

The MADSUM system is subject to misunderstandings if the user is not familiar with the concept of relative preferences. All preferences within the utility function are relative to one another, and so setting all sliders high, for instance, does not give the system any guidance. Also, setting improper constraints can prevent the system from responding as a naive user might expect. For example, setting a hard constraint for LENGTH of ten words and then setting high sliders for all three topics will not result in a ten word answer with three topics, since most single topic phrases are longer than ten words.

5.4 Multi-Attribute Utility-Based Evaluation

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. Utility is a number calculated for use 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. These are briefly defined here, then further explained in Sections 5.4.1–5.4.4.

$$Utility = \sum_{i=1}^{n} w_i f_i(a_{value_i}) \tag{5.2}$$

w_i Weights w_i, giving the importance of each attribute to the user, are calculated from the positions of sliders that are manipulated by the user in a graphical user interface.

a_{value_i} For resource attributes such as length of response or processing time, a_{value_i} is the actual value of the attribute, but for topic attributes a_{value_i} captures a numeric representation of significance to the decision at hand.

f_i Each of the functions f_i that appear in the utility function map their parameter a_{value_i} into a utility value between 0 and 1. The particular function f_i that is used

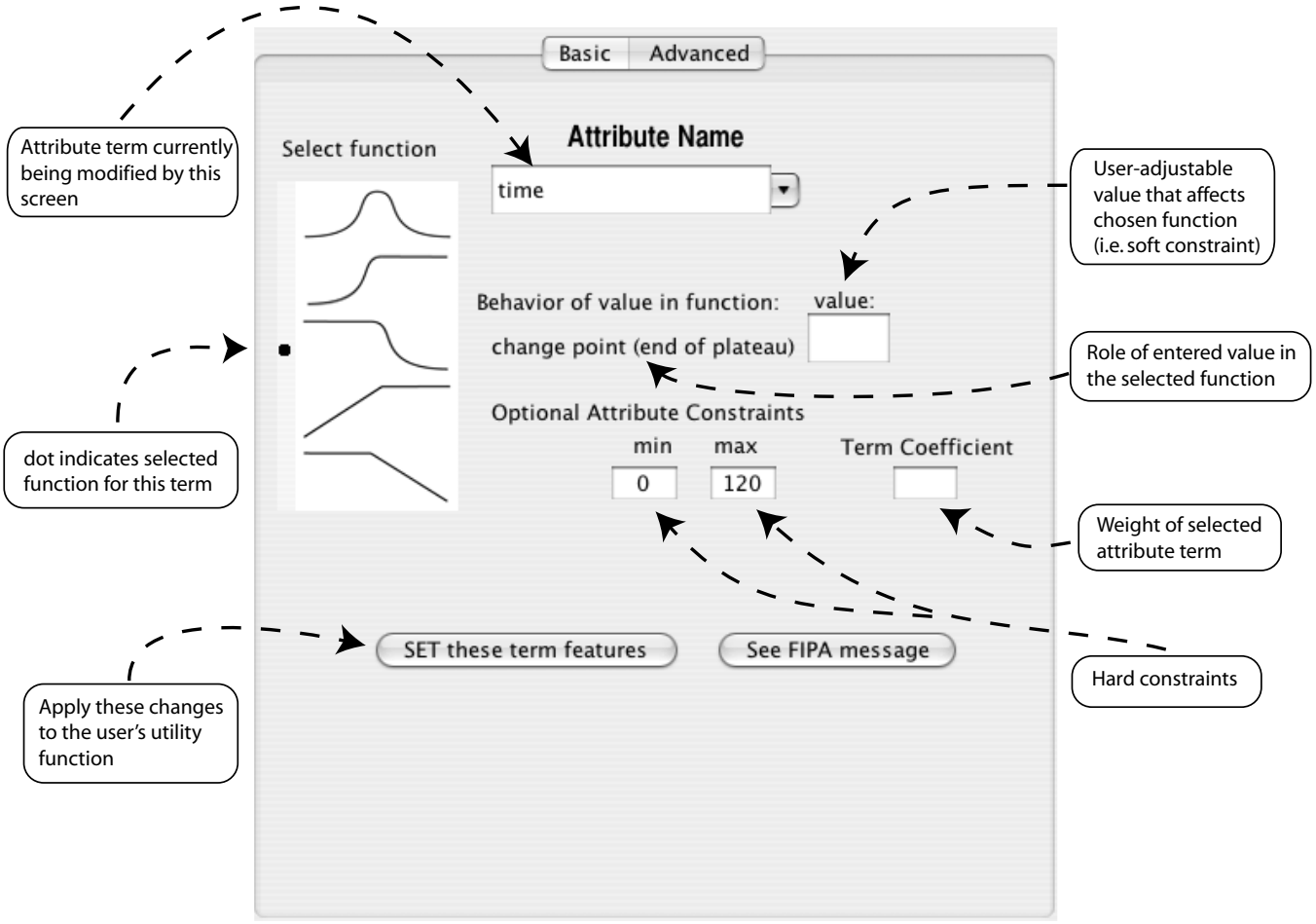


Figure 5.1: The advanced MADSUM interface, with the left column showing five available pre-defined functions (f_i) that can affect one term of the utility function.

determines whether an increasing parameter value increases or decreases utility (and at what rate). The ability to choose this function is one feature that distinguishes MADSUM from other decision-theoretic decision support systems, such as FLIGHTS[MFLW04] and MATCH[JBV$^+$], in which each a_{value_i} is treated in a fixed, system-determined manner.

5.4.1 The attribute weights w_i

An attribute weight is associated with each term of the user's utility function. Each weight can be set by the user via the sliders in the user interface (see Section 4.3)

to indicate the relative preference the user has for that attribute. For example, a user can set the RISK slider high to indicate that, other things being equal, they would prefer the final message to contain more *information about* risk. Note that this is very different from having the user state how they feel about risk as an investment parameter. By stating an interest in risk *information*, the user is not making any statement about how risky they want their investments to be.

The slider for each preference covers a one to ten continuum. The output of the slider is included in a message to the Presentation Agent (see 5.6) that starts the decision support process. MADSUM uses the attribute weight calculated from the slider via the formula

$$weight_i = sliderOutput * \frac{100}{\sum_{k=1}^{n} weight_k}$$

as a coefficient for that attribute term.

5.4.2 The functions f_i

Figure 5.1 illustrates the currently implemented functions in MADSUM's predefined library of utility functions. All functions map an attribute value into a $0..1$ valued space. Each function takes an attribute value as argument, but can also take an additional argument, which the user can set in the user interface. It is called "target value" on the basic screen, but is more accurately described in the advanced screen where the user can choose among different functions. The effect of the additional argument value varies according to the function, and is noted in the function descriptions below. If the user does not choose a value for the additional argument, it has a default value appropriate for the term (explained in the following section, Section 5.4.3). The functions are listed in the order that they are pictured under "Select functions" in Figure 5.1.

1. f_{Normal} approximates utility as a normal distribution of the possible values of its parameter. The optional additional parameter specifies the center of the distribution and the spread. This function would be appropriate if the user wanted to identify

a target value (e.g. LENGTH $= 75$) and have utility decrease slightly if the actual value were slightly different (e.g. 72 or 78) in either direction. The shape of the curve causes values that deviate significantly from the target to have significantly lower utility. The calculation is as follows:

$$f(a_{value_i}, target) = e^{-\frac{(a_{value_i} - target)^2}{2(target/5)^2}}$$

where $target/5$ affects the spread of the distribution in place of the standard deviation that would usually appear in the equation.

2. $f_{EndPlateauNorm}$ captures instances in which utility rises along a normal distribution curve for increasing values of its parameter until the parameter reaches the argument value, after which point the utility remains constant. The additional argument locates the starting point (left end) of the plateau and specifies the spread of the remaining curve. This function allows the user to declare a preference that utility rise until the argument value is reached, and be close to full utility when values are close to the value specified. For example, a user who believes that messages costing less than about 5 units are inaccurate might wish this behavior, or a user who wants a message longer than 100 words (e.g. to fill column space) but doesn't want to rule out a shorter message that is otherwise attractive.

3. $f_{StartPlateauNorm}$ captures instances in which utility remains high over a plateau and then decreases for increasing values of its parameter; the additional argument locates the rightmost endpoint of the plateau and specifies the spread of the remaining curve. This is the default function for resource attributes COST, LENGTH, and TIME, reflecting the idea that if a user specifies a cost, then costs which are lower should not be penalized, while costs that exceed the specified value should have utility decrease for increasing distance.

4. $f_{EndPlateauLinear}$ captures instances in which utility increases linearly for increasing values of its parameter until a plateau is reached at the change point specified by

the additional argument. This is the default function for information attributes (information significance values for RISK, VALUE, and GOAL). The function is linear to reflect the idea that the difference between values 1 and 2 is about the same as the difference between values 2 and 3, or 4 and 5. It reaches a plateau to indicate that values exceeding the optional argument are approximately equivalent.

5. $f_{StartPlateauLinear}$ is as above, except with the plateau at the beginning. The change point specifies the break between the plateau and the decreasing linear function; the linear portion meets the X-axis at $x = 2 * changePoint$. This would be useful if the user wanted to express that cost up to a change point was all of equal utility (for example, if an expense account would pay up to $10) but that additional cost was linearly undesirable.

My financial investment domain by default uses $f_{EndPlateauLinear}$ for information attributes, and $f_{StartPlateauNorm}$ for resource attributes.

5.4.3 Setting soft constraints

After selecting the appropriate f_i the user specifies soft constraints by using the additional argument to determine where the f_i changes shape. When an f_i is selected, the user interface displays text that explains how the additional argument relates to the f_i in question. In Figure 5.1 the selected function $f_{StartPlateauNorm}$ shows the text "change point (end of plateau)". The function $f_{StartPlateauNorm}$ is used by default for the resource attribute of processing TIME; the soft limit determines where the plateau ends and also the rate of fall in utility after the plateau (the falling portion resembles a normal distribution whose spread is 1/5 the soft limit). This captures the notions that 1) the soft limit on processing time set by the user is the point at which the utility of the response will begin to decrease and 2) the larger the soft limit on processing time, the less severe will be the loss of utility for each second of increased processing time.

Alternatively, instead of having the utility of length remain the same until the soft limit is reached, the user might want to say that utility increases with the length of the response until the soft limit is reached, and then decreases beyond that point; such a user might select the base function f_{Normal} to evaluate the contribution of length to the overall utility of the response. MADSUM's library of base utility functions is easily extendable should additional behaviors be desired.

If the user does not specify a value for the additional argument (soft constraint) to the utility function term's f_i, the system uses default values. For information attributes, the default value is 10. This value must be communicated to agent programmers so that they design agents whose information attribute values will all have the same range, making the values from different agents comparable. Resource attribute value defaults are as follows: for the length attribute, the default is 100 (words); for time, 60 (seconds); and for cost, 10 (units). These values are close to the system's current limits for resource consumption, and are chosen to allow the system to present a complex message by default.

5.4.4 Attribute values a_{value_i}

MADSUM is designed to accept an arbitrary number of attribute preferences, but for my implementation I selected the meta-attributes of text length (LENGTH), dollar cost (COST), and processing time (TIME), and three domain specific information attributes, one for each of three topics in the financial investment domain (see 4.2). These topics are RISK, VALUE, and GOAL. With these topic attributes the user can indicate his or her preference for information of a certain kind within the broader category of information about the investment under consideration. Together these six dimensions (the descriptive attributes LENGTH, COST, and TIME, and the information attributes RISK, VALUE, and GOAL) are referred to as *message attributes*. Each is captured by a separate term in the utility function, and so the user can indicate a preferences about each by modifying the term weight, function, and soft constraints as described above.

An attribute value a_{value_i} in the utility function is either a hard data value, as for LENGTH, COST, and TIME, or a number representing significance for the information attributes. Attributes capture characteristics of propositions that might be presented to the user, and for information significance the value captures the significance of a set of propositions in the environment of the user's personal characteristics and the application domain. I have termed 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 I have implemented such domain-specific information agents within three categories of information: RISK, VALUE, and GOAL. The associated decision specificity functions produce estimates of significance that are referred to as DS_r, DS_v, and DS_g respectively. It is important to note that DS corresponds not with the precise attribute value but instead with the significance of the information. In the financial investment domain, for example, the significance of a company's debt-to-equity ratio depends on its absolute debt-to-equity ratio (heavy debt is bad) and also possibly on the company's industry classification (high debt is more acceptable in utilities than in manufacturing). Thus a .8 debt-to-equity ratio would be noteworthy in some instances but not in others. (MADSUM source agents offer two-tier pricing for information, with the absolute data at low cost and the data in context of the industry at slightly higher cost.) For example, suppose a source agent, DebtEquity, obtains 0.6 as the debt-to-equity ratio for company XYZ, and 0.8 as the industry average debt-to-equity ratio, and sends it to the task agent (its wrapper) DebtRisk. DebtRisk uses a domain-specific and ratio-specific formula[1] to calculate how good or bad XYZ's ratio is in light of the industry

[1] The particular equations used by each domain agent to determine DS are not important to my work. Financial analysis texts I examined only associated specific ratio values with verbal descriptions of their import, so I extrapolated numeric formulas from textual descriptions in [Bra02].

ratio. In this case, the value of 0.6 means that for a company of its size[2] XYZ has substantially lower debt than is average in its industry. DebtRisk expresses the significance of the result of this comparison by assigning a DS of 4.83. This is a very significant level of DS, indicating that the information (about the *relative* value of XYZ's debt-to-equity ratio and the industry average ratio) is likely to be of significant value to the user in making a decision about an investment in XYZ. Similarly, the significance of a proposition from the Portfolio Agent that addresses the relationship of a proposed investment to the user's portfolio allocation goals depends on the extent that the investment would cause the user's portfolio allocation to deviate from his goals, while the DS of a proposition that addresses the appropriateness of the investment from an age perspective may depend on how close the user is to retirement. The DS value reported by a domain expert agent reflects the thinking of the designer of that agent; thus agent designers must be careful to agree on consistent meanings for DS values to ensure that values can be meaningfully compared across topics.

For content selection in MADSUM, the DS of a set of propositions in a particular category (RISK, VALUE, or GOAL) is computed as the sum of the DS values for the individual propositions; this has worked well in my application for content selection but further experimentation is needed to fully validate the decision.

DS is similar to the *s-compellingness* measure adapted by [WWS+04, MFLW04] from [KS94] to estimate the significance of a single attribute value to a user (i.e. how compelling it would be as an argument supporting the system's recommendation). S-compellingness is the weight given to an attribute in the utility function times the maximum of the two numeric "distances": the distance between the attribute value and the best possible value it could have, and the distance between the attribute value and the worst possible value. Intuitively, attributes that are close to the extremes of their value range are more likely to make persuasive arguments, either for or against a recommendation.

[2] Actually, for a company of its level of capitalization...

DS differs from s-compellingness in two ways. First, DS is assigned to a proposition by a domain expert agent. Thus the decision-specificity of a particular value of the Debt-Equity ratio is determined by the Debt-Equity Agent. This enables expert agents to change what they consider significant over time. Second, DS does not include the user weight applied to the attribute; instead the weight is applied when utility is calculated. This is essential since the user is allowed to change their utility function during execution. The recorded DS numbers of a proposition or text plan tree remain stable while the weights in the utility function can fluctuate.

5.4.5 The complete utility function

Thus the magnitude of a single term in the complete 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 which is typically selected by MADSUM but which can be selected by expert users (under the advanced features of the user interface), and two user-selected modifiers (the weight w_i that gives the importance of this attribute to the user, and the additional argument that adapts the function f_i to reflect soft constraints). 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.

5.5 Assembling Text Plans

Chapter 6 discusses the agent-based architecture in which my system collects and integrates information from distributed sources. In this section, I discuss the rules that are used to organize individual pieces of text into a coherent framework, ignoring for the moment the agent architecture. The coherence rules are based upon work by other researchers in natural language generation and are not the focus of my work.

5.5.1 Related work

Like RTPI described in Section 2.6, the theory that underlies MADSUM's text plan assembly is Rhetorical Structure Theory (RST)[MT83, MT87]. However, the text plan assembly rules in MADSUM do not modify and rearrange the internal structures of the subtrees to the same degree as those in RTPI (see 5.5.2). This is the result of a fundamental difference in the systems. RTPI was part of a system in which it had full knowledge of the semantics underlying each part of every text plan, and so it could have rules that used that knowledge to safely rearrange the subtrees it received as input. In contrast, MADSUM must integrate text plan trees without detailed knowledge of the contents, by superimposing structure rather than changing it.

The MADSUM system can be viewed as a bottom-up text planning process distributed across multiple agents, where small text plans created by wrapper agents near the bottom of the hierarchy are combined by the agents above them until a single plan is created/selected at the top. ILEX [MOOK98] is a bottom-up, opportunistic RST-based planner implemented in a single program. ILEX and MADSUM are similar in that both superimpose connecting relations on pairs of subtrees; however, because ILEX is not distributed it can consider all the possible combinations of all possible subtrees at once. In fact, ILEX employs various heuristics and a genetic algorithm to manage the complexity involved. MADSUM, having the sources of the subtrees distributed, avoids the same level of complexity (but also cannot consider as many possible different combinations). Both ILEX and MADSUM return result trees that may not be optimal. Like MADSUM, ILEX can achieve a certain text length; however, it does not do so during planning, but by pruning a completed plan at the end of the planning stage.

Marcu's bottom-up approach to text planning [Mar97a] was the first to be designed to express all of the content about a given topic that was present in the knowledge base. Similarly, a MADSUM agent presented with a set of text plan trees must design a new tree incorporating all of them. However, Marcu's system is incorporating content propositions

(with a known set of RST relations existing between them), not separate text plan trees.

5.5.2 Generating Multi-Sentence Text

MADSUM is currently implemented with three simple rules for combining pairs of text plan trees. Text plan trees in MADSUM consist of nodes that represent RST-style relations that hold between the children (see [Moo95] for a thorough explanation). For example, a tree with a PERSUADE node at its root might have an INFORM leaf node and a MOTIVATE node as children; the interpretation of the tree is that the system wants the user to do/believe what is in the INFORM node, and intends to persuade the user by using the strategy contained in the tree rooted at the MOTIVATE node.

Each tree structure also contains a numeric representation of DS, or decision specificity. The DS value of a text plan tree is stored as two sets of three numbers, as in:

$$[[2,\ 2,\ 1][2.5,\ 4,\ 2]]$$

The first triplet represents the DS of parts of the text plan that support the buyer's purchase of the investment, while the second triplet refers to parts of the plan that do not support the purchase. For convenience, "positive" DS will refer to the left triplet and "negative" DS will refer to the right triplet. Within each triplet the first number represents the the highest (or "peak") DS of any individual proposition in the text plan; the second number represents the sum of DS for all the parts of the text plan; and the third (an integer) represents the number of parts in the plan. Storing these numbers allows the system to consider peak or sum DS in either direction[3]. Peak DS is used when computing utility for text plan ordering, thus reducing the likelihood that a large number of insignificant propositions would have precedence over a single highly significant proposition.

[3] Currently the system only uses the parts count to identify singleton trees (which have a count of one in exactly one direction), though the count could be used if later research showed that tree size or average DS were somehow useful in integration or realization.

When text plan trees are integrated, DS requires a special accumulation function (a sound way of putting two DS representations together). Almost all systems that use utility functions assume that the information in all terms is additive[KS94, WWS+04]. For example, LENGTH is additive in *MADSUM*, so that if two propositions of length 20 and 30 words, respectively, are joined, then the utility of the combination will be based on a length of $20 + 30 = 50$. However, this is not true for the *MADSUM* attribute TIME. If two results are expected to take 20 and 30 seconds, respectively, then the combined result is expected to take 30 seconds, since *MADSUM* assumes its agents are operating on different machines. Thus the accumulation function for TIME is the max function. Since DS for a proposition or tree is represented as a pair of triplets, it is clearly not additive. Instead, DS has a special accumulation function (shown as $\oplus$) that operates on the triplet pairs as described above, such that

$$\begin{aligned}
&[[x^+_{max}\ x^+_{sum}\ x^+_{count}][x^-_{max}\ x^-_{sum}\ x^-_{count}]] \ \oplus \\
&[[y^+_{max}\ y^+_{sum}\ y^+_{count}][y^-_{max}\ y^-_{sum}\ y^-_{count}]] \quad = \\
&[[max(x^+_{max}, y^+_{max})\ sum(x^+_{sum}, y^+_{sum})\ sum(x^+_{count}, y^+_{count})] \\
&\ [max(x^-_{max}, y^-_{max})\ sum(x^-_{sum}, y^-_{sum})\ sum(x^-_{count}, y^-_{count})]]
\end{aligned} \tag{5.3}$$

as demonstrated in these examples:

$$\begin{aligned}
[[1\ 1\ 1][0\ 0\ 0]] \oplus [[2.99\ 2.99\ 1][0\ 0\ 0]] &= [[2.99\ 3.99\ 2][0\ 0\ 0]] \\
&and \\
[[2.99\ 3.99\ 2][0\ 0\ 0]] \oplus [[0\ 0\ 0][0.5\ 0.5\ 1]] &= [[2.99\ 3.99\ 2][0.5\ 0.5\ 1]]
\end{aligned} \tag{5.4}$$

thus preserving peak, sum, and count values for both positive and negative DS.

5.5.2.1 Combining text plan trees

MADSUM uses a set of domain-independent rules for combining all component trees into one. I will describe how the rules are applied, and then review the rules themselves.

Once a set of text plans is presented to an agent from agents below, the agent is required to design a text plan tree incorporating all of the received component trees. The set is first ordered by a reduced version of the utility function. Since resources such as overall text LENGTH, COST, TIME are not affected by the final order of the text parts, and since the users' preferences about these resources cannot be construed to prefer pieces of text over one another, the altered utility function excludes the resources length, cost, and time, and is calculated on the information attributes (peak DS for RISK, VALUE, and GOAL). Individual integration rules order components in the resulting tree according to the same reduced version of the utility function, again emphasizing peak DS (the highest DS in any subtree), with the highest peak-DS-utility subtree always placed on the left side of the tree; this ensures that highly significant information will appear before a large subtree containing many pieces of low significance information, other features being equal. Since rules are applied to combinations of trees from the bottom up, the resulting tree and text will have the highest (reduced feature) utility text fragment first. If the user's coefficients for information attributes are all the same, then the highest peak DS text fragment will be first.

This seems counter-intuitive at first; the component trees were chosen based on the *total* utility they provide, so why not order them the same way? The answer lies in the diverse nature of the utility components. Utility for a very uninteresting component tree t_{long} may only be high because the associated text will be very long and the user placed a high priority (coefficient) on LENGTH. Another component tree, t_{short}, has a very high DS (and so will be significant to the user) but a lower overall utility than t_{long} because the user emphasized LENGTH. I hypothesized that even when a user wanted a long overall response, there was nothing about the length of a single component that should cause it to appear before another, and so the shorter, more significant component should appear first in the resulting message. This was confirmed by limited testing and should be further explored.

Yet another option would be to order the subtrees according to a utility function using sum DS instead of peak DS. Consider combining two trees, t_1 and t_2. Tree t_1 has three subtrees, each with a sum DS of 4; thus the sum DS of t_1 is 12. In contrast, t_2 is a singleton tree and has a peak and sum DS of 10. If the trees are ordered by sum DS, the most significant single piece of information (in t_2) would be far to the right side of the tree, to be realized last, which violates my ordering hypothesis.

Note that MADSUM is not ordering sentences, but text plans submitted by agents. Since an agent submitting a plan to the Presentation Agent may have contracted with many layers of agents, the text plan it submits could be multi-sentential. The ordering of information within that plan is done *by the submitting agent*, and the information is not re-ordered by the Presentation Agent (or any other agent in the hierarchy). Thus all the text sub-plans regarding the topic of *risk* are ordered by the RISK agent, so that all risk-related information will be together in the final plan.

Barzilay et al [BEM01] performed experiments in sentence ordering within a paragraph when a system has multiple sources of information. The multiple sources are not supplying discrete information like the agents of MADSUM, but rather produce duplicate, or almost duplicate information. Barzilay et al show sub-optimal results for two kinds of ordering: majority rule, where the information duplicated by the most sources (and therefore possibly the most significant or most reliable) comes first; and temporal, where ordering reflects some time value intrinsic in the information (such as a sequence of events). But their best result was obtained when they used information *theme* (similar to topic) to group sentences (as done in MADSUM).

5.5.2.2 Text plan integration rules in *MADSUM*

The three rules are applied in the order below to each pair of trees. These rules happen to be exclusive in their application, i.e. only one of them will apply to any given pair of trees, but the system does not rely on this property. As new trees are created they are added to the set, and the component trees are not removed, so that the set consists of

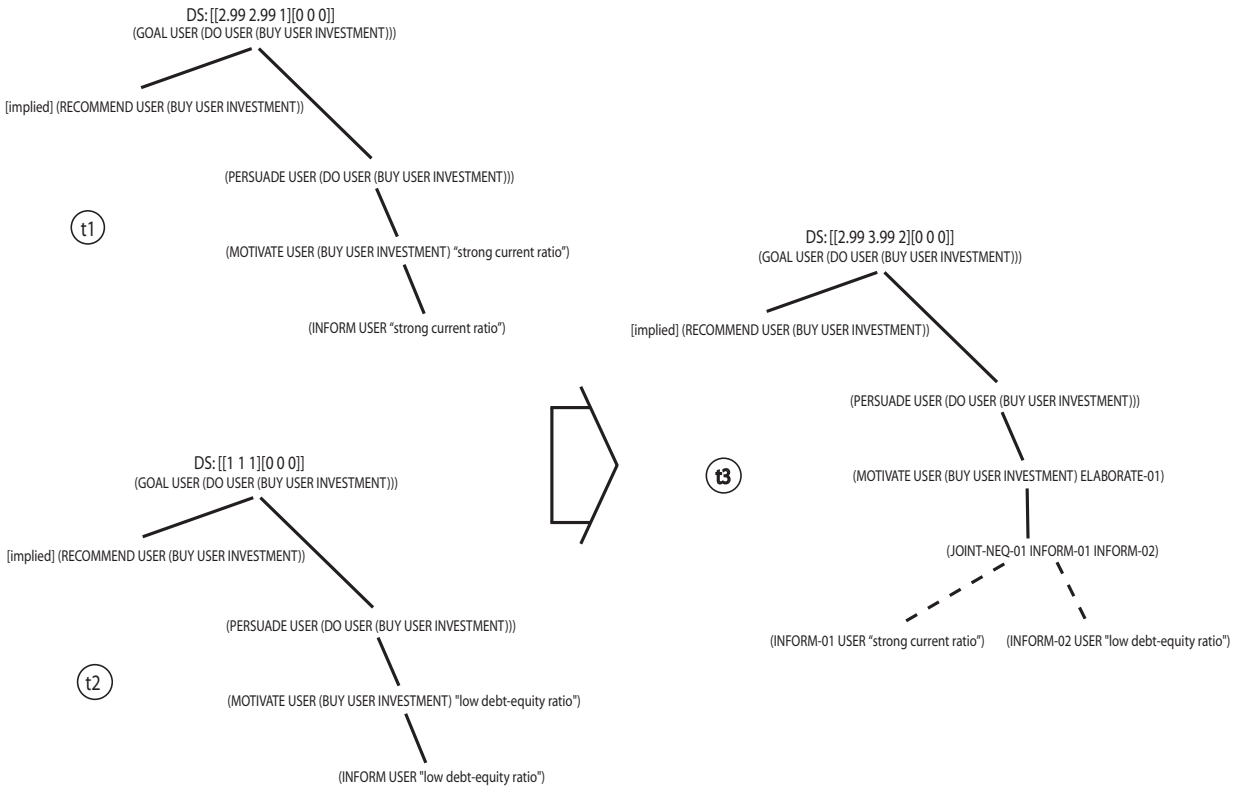


Figure 5.2: Two simple text plans (t1 and t2), and the result (dotted lines) of applying the Joint-Under-NEQ rule (t3).

both small component trees and partially or fully assembled trees. Leaving the component trees in the set allows the components to be assembled using different rules, as well as the comparison of trees built by rules applied in different orders.

1. The Joint-Under rule (Figure 5.2) is designed to aggregate the supporting clauses of two trees that are trying to accomplish the same purpose (this is determined by comparing the structure of the trees over the clauses under consideration). It is only appropriate if the two trees are of the same topic and DS orientation. When the two supporting clauses are of the same or similar DS, the relation is labeled "Joint". However, if one supporting clause is substantially more significant (higher DS by one or more) than the other, the relation is labeled Joint-Under-NEQ. The NEQ

in the name stands for "not equal" and refers to the difference in the DS, while "under" refers to the fact that the effect of the rule is to create a new tree with the same superstructure and modify what is "under"it, as opposed to superimposing a new superstructure over two trees.

In Figure 5.2 both t1 and t2 have the same DS orientation, and the difference is $2.99 - 1 = 1.99$, which is greater than one, so the Joint-Under-NEQ may apply. Now the structures of the trees are compared. When the trees are found to share the same GOAL-PERSUADE-MOTIVATE sequence, but with two different propositions ("strong current ratio" vs. "low debt-equity ratio"), the rule fires. It creates a copy of the top of the first tree, and creates a new JOINT-NEQ node with both propositions underneath. The new DS structure in t3 reflects that peak DS is 2.99, sum is 3.99, and t3 contains two original subtrees.

Figure 5.3: Two plans (t3 from Figure 5.2 and a new tree t4 shown here as a subtree), and the result (dotted lines) of applying the Contrast rule (t5).

2. When two trees do not agree on an investment (i.e. the DS orientations are different) then the Contrast rule can be applied, even if the trees are not of the same topic (Figure 5.3). If the peak DS of the two trees is similar, they will be joined under a Contrast relation. Trees with dissimilar peak DS values will be joined under Concession relations, since a concession implies that one tree is more significant to the message than the other. However, all the DS values of each tree must be in exactly one direction, i.e. one tree must have all positive DS and the other all negative; thus a Contrast will not be superimposed over another Contrast or Concession relation. This is because it is very difficult (even for humans) to unambiguously express nested contrasts or concessions.

   Figure 5.3 shows that trees t3 and t4 have opposite DS orientation, so this rule applies. Because $|2.99 - 0.5| = 2.49$, which is greater than 1, the rule will fire and create a CONCESSION node to superimpose over copies of the two subtrees.

3. Finally, when two trees cannot be joined any other way, a Joint relation can be superimposed *over* them (this is in contrast to the "joint" rules above where joints are used to aggregate information under a common tree structure). This is avoided if possible since a Joint contains no useful semantic content to aid realization (or reader understanding).

More than one rule may apply to any pair of subtrees, resulting in multiple integrated trees from the same set of subtrees. As various integrated trees are generated by applying rules, they are maintained in a list ordered by utility, calculated under the current utility function. If the agent reaches a timeout, it simply returns the highest utility tree found so far. If the agent does not run out of time, it will eventually construct all possible combinations of the available subtrees using the rules available. For example, two subtrees with different DS orientations may be combined using both a Contrast rule and a Joint rule. Given sufficient time, constructing both trees allows the consideration of other ways to combine the integrated tree with yet other sub-trees. When trees differ

in the relations used to integrate them, trees assembled with Contrast or Concession relations are preferred over trees that are assembled with Joint-Under rules, which in turn are preferred over simple Joint relations. Using a hierarchy to value one kind of relation over another has been done by other bottom-up text planners [Mar97a, MOOK98]. If two trees still appear identical, then one is chosen randomly as the high utility tree.

5.5.2.3 Generating Natural Language Via Templates

MADSUM generates text in a three stage process:

1. Domain specific agents provide the template-based text for the leaf nodes of each tree fragment they create, and insert the text into a text plan tree template.

2. Text plan trees are propagated up the agent hierarchy, and as trees are assembled/integrated, rule applications can modify the tree structure to affect realization (e.g. the aggregation performed by the Joint-Under-NEQ rule).

3. When the Presentation agent finishes creating a single tree from the trees it receives from below, the agent provides domain independent connectives and punctuation based on the full tree structure.

Templates offer many advantages: rapid development, simplicity, and the ability to include some complex grammatical structures without deep analysis of the text plan trees and domain concepts. [Rei95]. There is also substantial development cost involved in using an existing syntactic realizer [Rei99]. Some recent systems also use templates [WWS$^+$04], while other adaptive systems employ full syntactic realizers [JBV$^+$, MFLW04], but working with templates allowed me to focus on the issues of content selection, limited aggregation, and ordering. MADSUMS's use of RST-style trees will facilitate a transition to a full syntactic realizer should that be desirable in the future.

5.6 Implementation

I have implemented and tested the MADSUM architecture for adaptive response generation in a financial investment domain. MADSUM is implemented in Java(tm), and has been tested on Sun workstations and Apple G4 desktops and laptops. For details on system performance, see Section 6.7.

MADSUM has agent "wrappers" whose task is to make text trees out of the data returned by information source agents. The source agents can derive their information from external data sources (e.g. websites or other agents), internal stored data from previous external acquisitions, or analysis of data. Currently all source agents access local data files to acquire information, to reduce the vagaries associated with dynamic information gathering (this allows my testing to focus on the dynamics of the data gathered, agent interactions, and user preferences). Work devoted to information gathering includes agents designed for a specific data gathering task and also automatically generated agents for simple data gathering [KAH94, Kus00].

5.7 Examples of Adaptive Responses

Consider a user who proposes the purchase of 100 shares of stock in IBM. The user model contains personal characteristics of the user, including her current investment portfolio and her portfolio allocation goals. In addition, before proposing the stock purchase, the user has set soft constraints on the length of the response, the cost in dollars of any purchased information, and processing time. She has also adjusted sliders on the graphical user interface to indicate the importance she assigns to usage of different resources (length of response, cost, and processing time) and her interest in information that addresses each of the different content categories (investment risk, value, and impact on portfolio allocation).

Figure 5.4 displays MADSUM's response under different soft constraint and priority settings. In Figure 5.4a, the soft constraint on length was 75 words and the user placed a higher priority on risk information than on value and portfolio information. For

5.4a: 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.

5.4b: 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.

5.4c: 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 5.4: Three responses, derived from different soft constraints and priority settings.

the responses in Figure 5.4b and Figure 5.4c, the soft constraint on length was lowered to 35 words, resulting in the exclusion of some available propositions. In addition, the relative priorities on risk, value, and portfolio information were kept the same in Figures 5.4a and 5.4b, but were altered in Figure 5.4c to place a much higher priority on value information than on risk or portfolio information. Due to the 35 word soft constraint on length that was set for the response in Figure 5.4b, propositions had to be excluded. Since risk was given highest priority, much (but not all) of the risk information was included. However, the high significance of the proposition about the impact of the proposed investment on the user's portfolio allocation goals (she had already exceeded her goals for equities such as IBM) caused that proposition to increase the estimated overall utility of a response containing this proposition, and thus it was included despite the length of the resulting response slightly exceeding the soft constraint on length. In Figure 5.4c, the user's

much higher priority for value information resulted in selection of the value proposition, even though it was of lesser significance than other available propositions. In addition, the highly significant proposition about portfolio allocation goals was included in the response. These examples illustrate the system's ability to vary its responses depending on the user's resource constraints, the significance of information, and the priority that the user assigns to different resources and kinds of information content.

5.8 Evaluation

I have implemented and tested the MADSUM architecture for adaptive response generation in a financial investment domain. The examples in Figure 5.4 are actual responses produced by the system under the conditions described in Section 5.7. Experiments have demonstrated the system's success at varying its responses to adapt to different resource constraints and different priorities for resource and content attributes. This section explains my choice of evaluation methodologies and presents a summary of the results of human subject evaluations.

Mellish and Dale [MD98], having investigated the subject of evaluating NLG systems, recommend 1) that the components of a system be evaluated separately to distinguish their individual performance, and 2) that evaluations by humans use ranking when possible, instead of open questions, to help unify the continuum of responses. They report that many systems that ask seemingly simple open questions end up with a lack of human agreement that is very hard to statistically overcome .

In keeping with those strategies, the response generation aspect of my research can be viewed as having two components, content selection and content expression. Of these two, content selection is fundamental to and inherent in the design of MADSUM, insofar as it is the product of the user's utility function and the agent negotiation process. It was critical, therefore, to evaluate MADSUM's content selection choices. With respect to content expression, I evaluated MADSUM's ordering of chosen content. While this aspect of MADSUM's response generation is not intrinsic to the system's design, it is

nevertheless an important implementation decision that will influence any future extensions of the system. Evaluation of MADSUM messages beyond content selection and ordering (e.g. keywords, phrasing, punctuation) should take place after a full syntactic realizer is in place.

5.8.1 Content selection testing

I tested content selection by having subjects indicate which of two sets of propositions they thought would best fit the user's information preferences, as indicated by a picture showing the sliders set by the user. The purpose of the experiment was to validate *MADSUM*'s content selection based on the value of a utility function that includes user preferences about both message attributes and information significance (DS). The actual questionnaire is in Appendix C.

In the experiment, twenty-one subjects were presented with seven scenarios consisting of 1) a graphic depiction of sliders representing user priorities for the three kinds of content (RISK, VALUE, and GOAL), and 2) two sets of propositions, one of which had been produced by the system. The alternative was generated by hand using the strategy *not* used by the system, i.e. if the system response content appeared to reflect user preference[4], then the alternative content was selected based on significance (DS). In some pairs, the significance (DS) of propositions and the priority that the user placed on that kind of information were congruent (each proposition was either both significant and high priority, or both of lesser significance and lesser priority). Propositions in both sets were ordered by significance. The system's response was listed first in some pairs and second in others.

There were four content selection questions (not placed consecutively) in which information significance (DS) and user priority were congruent. Twenty-one people examined four pairs of this kind, for a total of 84 tests, and the scores for these questions

[4] The system does not choose to reflect either user preference or information significance (DS), but considers both in a utility function.

were as follows:

scenario	1	2	3	4	
number correct	19	20	21	18	total: $78/84$

A one-tailed binomial test shows that this result is highly significant ($p << .01$).

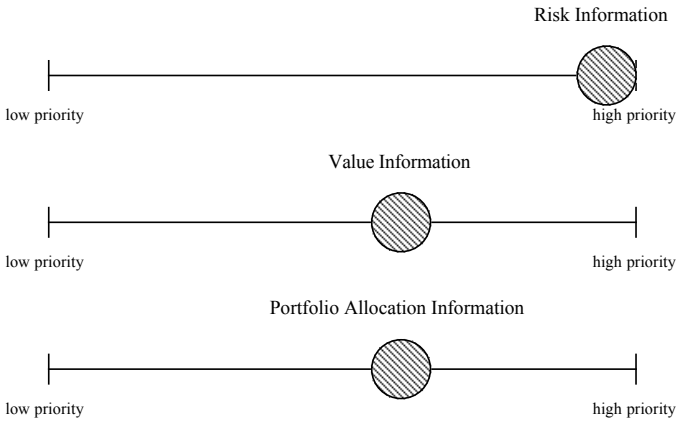
In three scenarios the user's priorities and information significance conflicted. For example, the user might indicate a strong preference for RISK information, but the proposition for VALUE appeared to be more significant than the proposition for RISK. In such a case, the user might decide that:

1. the system should select information only based on apparent significance; or
2. the system should base content selection only on what information preferences are indicated by the user's sliders; or
3. some balance of significance and user priority should be used to affect content selection.

MADSUM implements the third option, by using a utility function that includes both information significance (DS) and user preferences. Subjects were given the opportunity to choose between responses that favored priority in content selection, responses that favored significance, and responses that balanced priority and significance.

An example scenario from the content selection evaluation is presented in Figure 5.5. In this example the user's priorities and information significance were in conflict. The user has set the sliders to show a preference for RISK information. However, in the two content sets shown (these sets both contain only a single proposition) the information about RISK appears much more significant to a reader than the VALUE information. The reader can distinguish which topics these ratios belong to because they have an instruction sheet (see Appendix B) that lists the financial ratios which correspond to each topic.

2. Taking into account **both** the priorities of the investor and the importance of the information, which set's contents will be most helpful to the investor making a decision?



(a) company has historically maintained a moderate debt policy

(b) company has a very poor price earnings ratio, suggesting it may be over-valued

Which set did you choose?

Briefly, why?

Figure 5.5: A sample content selection evaluation scenario.

Twenty-one subjects were asked three questions for which significance was not congruent with user priority, for a total of 62 tests, and the results were:

scenario	1	2	3	
number correct	19	11	19	total: $49/62$

A one-tailed binomial test shows that this result is highly significant ($p << .01$).

The significance of the results for the content selection scenarios indicates that the subjects did, indeed, think that the system should take into account both information significance (DS) and user preferences, and further, that the subjects agreed (to a significant degree) with the particular content selection choices made by *MADSUM* for the examples provided.

5.8.2 Content order testing

As discussed in Section 5.5.2, *MADSUM* places the highest utility propositions at the "beginning" (to the left) of each integrated text plan tree, but only after the length, cost, and time terms have been removed from the utility function. This was done because I hypothesized that once content was selected, the user's preferences about the message's entire length, cost and time should not affect the ordering of text within the message. Thus utility for ordering consists of the DS terms, with a user specified coefficient and peak DS as determined by the agent that generated that portion of the text plan. As trees are recursively integrated this results in the subtree that contains the proposition with the highest peak-DS utility being placed to the left, where it will be realized first. To test the system's ordering of propositions in its presentation to the user, I 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. The subjects were presented with two different orders of presentation of the same propositions (i.e. the content was the same in both sets). In each case, most cue phrases and connectives were removed in an attempt to prevent subjects from being influenced by phrasing.

Four of the nine scenarios (not presented consecutively) had propositions where the most important proposition was of the same topic that the user had indicated was the highest priority, i.e. the significance and user priority were congruent. For these scenarios the results were as follows:

scenario	1	2	3	4	
number correct	15	14	15	15	total: $59/64$

A one-tailed binomial test shows that this result is highly significant ($p << .01$).

The remaining five scenarios consisted of sets where the most significant proposition was not congruent with the user's information preferences. Thus the subjects had to decide whether the system should order information:

- based only on apparent significance; or
- based only on what information preferences are indicated by the user's sliders; or
- according to some balance of significance and user priority.

The results for the 16 subjects on five non-congruent ordering questions (not presented consecutively) were:

scenario	1	2	3	4	5	
number correct	6	10	12	10	9	total: $47/80$

A one-tailed binomial test showed that this result was significant ($p < .05$).

All ordering tests combined yielded results of 106 correct (i.e. subject chose the system result) out of 144 tests. A one-tailed binomial test shows that this result is highly significant ($p << .01$). This result supports *MADSUM*'s method of ordering propositions by showing that subjects agree with selection and ordering choices made by *MADSUM*. The results for individual questions also suggest that system ordering performance may have room for improvement when user priorities and significance are not congruent.

One must note that full evaluation of a decision support system requires that subjects interact with the system on a real decision that they want to make. Only then is the user in the "frame of mind" such that he or she can make a reliable judgement about the system's performance. Nonetheless, these evaluation experiments support the content

selection and ordering strategies used by MADSUM and suggest that they will contribute to producing quality response generation in decision support.

5.9 Summary

MADSUM produces responses that are tailored to a particular user. The user can explicitly influence the system behavior by setting preferences on attributes of the response, including topic priorities and response COST, LENGTH, and processing TIME. The user can also constrain the system's use of available resources in a hard or soft manner. I have demonstrated that even under a consistent data environment, MADSUM responds differently under different user preferences. Furthermore, my evaluations show that subjects strongly concur with MADSUM's choice of content, and concur to a lesser (but still significant) degree with MADSUM's ordering of content.

Chapter 6 details the contribution of the architecture to the response, as well as the features that allow MADSUM to operate successfully in a dynamic environment.

Chapter 6

AN AGENT-BASED ARCHITECTURE

Information of all kinds is increasingly available from distributed sources. Decision-support systems typically use information from a broad range of sources; such a system might have access to a variety of expert information providers as well as generic news sources. The number and variety of sources currently available freely on the Web creates tremendous opportunities for a system that can exploit them to the advantage of a particular user. (*MADSUM* agents can take advantage of free information sources, but *MADSUM* is designed to weigh cost as part of its utility calculation.)

However, the availability of multiple sources provides computational challenges in addition to opportunities. Problems that arise for the system include determining:

1. which sources to use: *MADSUM* sources will differ in the type of information they provide, the quality of the information, and pricing. Other source considerations could include communication quality, consistency, discounting, collaborative behavior, etc.

2. how to allocate finite resources across sources: in the investment domain, *MADSUM* considers the resources of COST (dollar cost), LENGTH (text length in words), and TIME (seconds from initial request to presentation of a message).

3. how to integrate results from different sources: The results in *MADSUM* will be text plan trees, which *MADSUM* integrates.

4. what to do when sources don't perform as agreed or results do not meet expectations; and

5. how to react when the environment changes (including resources, other agents, or the user).

To be a successful decision-support system, *MADSUM* has to address all of these issues to some degree. Multi-agent systems are commonly used as a means of addressing the problems and opportunities presented by a dynamic, heterogeneous information environment[Klu01], and so I chose to implement *MADSUM* as a system of multiple software agents.

In the previous chapter, I presented a multi-attribute decision-theoretic approach to generating responses in a decision support system. I have implemented this approach within an agent-based architecture that addresses the above issues and enables flexible, fail-soft behavior. This chapter presents the agent-based framework. Section 6.1 discusses related architecture research, and Section 6.2 outlines the DECAF agent architecture which underlies *MADSUM*. Section 6.3 describes the *MADSUM* architecture. The agent negotiation process is presented in Section 6.4, and the resulting execution is presented in Section 6.5. *MADSUM*'s failure handling protocol is described in Section 6.6, and Section 6.7 presents empirical results supporting earlier claims made about *MADSUM*'s performance. The chapter is summarized in Section 6.8.

6.1 Related work

Research related to the *MADSUM* architecture has been done in the areas of decision support, agents using utility theory, resource allocation and negotiation. The decision support system BIG [LHK+98] has an architecture which shares many of the features of *MADSUM*, as well as having many features *MADSUM* does not have. BIG locates/discovers, retrieves, and processes information to help a user make a decision about

purchasing a software package. BIG was designed as a next-generation information system that would integrate numerous artificial intelligence technologies: scheduling, planning, text processing, information retrieval and extraction, and limited problem solving.

There are also many differences between the systems. First, while *MADSUM* is composed of multiple software agents, BIG is a single, highly complex information gathering agent. Second, BIG displays results as a series of product feature name/value pairs organized in a form, while *MADSUM* produces text. Third, *MADSUM* allocates resources based on a decision-theoretic measure of various user preferences, while BIG does not, as explained below.

Important characteristics shared by *MADSUM* and BIG are that both are designed to use the TÆMS formalism (see Section 6.2) to express the trade-offs between cost, quality, and duration that are required for intelligent scheduling. While *MADSUM* explicitly represents the information required by TÆMS, *MADSUM* has not fully implemented the TÆMS interface in its underlying architecture, DECAF (see Section 6.2), while BIG implements it and is investigating design issues. In particular, BIG uses TÆMS to reason about uncertainty, and opportunistically plans to address uncertainty. This would be very applicable to the problem *MADSUM* addresses if the uncertainty were related to the information that BIG gathers; however, the uncertainty in question is not about information, but rather the end-to-end performance of the system.

As in *MADSUM*, the user can also place hard constraints on the amount of time and money BIG will spend in developing an answer. BIG does allow the user to specify one meta-preference about the information result it produces (as opposed to the preferences the user expresses about the software under consideration). The user can indicate a preference for information precision (increased detail) versus breadth of coverage (number of sources explored), which will influence how BIG chooses to allocate resources. A number representing the precision/coverage trade-off is used directly as the quality metric by the scheduler. This is in contrast to *MADSUM*'s arbitrarily-sized vector of user

preference attributes, each associated with a weight, which can then be combined to form a single overall utility measure. Thus *MADSUM* uses a decision theoretic means to allocate resources based on multiple user preferences about the message, while BIG uses multiple uses preferences about the *software product* being considered, and a single user preference about the message in conjunction with cost and time constraints.

Like *MADSUM*, BIG must make choices about how to integrate information from different sources. In the case of BIG, the sources are not different agents within the system, but actual web or database information found by BIG. In the BIG system, the process is called "information fusion", and it assigns an ordinal information quality number to each piece of information it discovers or calculates, based on some confidence assigned to the information type or source. The actual information is then weighted according to the assigned quality. The disadvantage of this approach is that information about a high product rating from a high-quality site, combined with information about a low product rating from a different high quality site, will result in a neutral combined rating. In contrast, *MADSUM* would present both pieces of information in a CONTRAST relation, thus indicating to the user not that the product is of middling quality, but rather that two sources disagree dramatically about the quality of the product.

BIG *is* decision theoretic in the same way as FLIGHTS [WWS+04] and MATCH [MFLW04], namely that BIG builds a model of each product that it discovers, based on a list of user-specified software features (not message attributes, but attributes of the product under investigation), and then calculates the utility of the model to the user - *not* the utility of the system's *response* to the user. As explained in Section 5.2.2.2, *MADSUM* pays attention to the utility of the *message* to the user, not the utility of a model outcome.

BIG is a highly complex agent focused on gathering and interpreting information. Typical run times for the full system are approximately thirty minutes. Since BIG gathers information and processes it into a form, it would be a simple matter to map the form into RST-style templates and use BIG as a powerful source agent for *MADSUM*. Thus

multiple BIG agents could focus on gathering and interpreting information on a single topic, and *MADSUM* could form a network of such agents and integrate their messages in a decision theoretic manner.

6.1.1 Resource allocation

Whenever multiple agents are involved in a task, the question of how resources should be allocated to individual agents arises. Other multi-agent systems have explored a variety of ways to allocate resources across agents contracting to perform subtasks. Techniques explored recently include reinforcement learning [GCL04], extensive communication until a time limit is reached [ZLP05], and swarm behavior [dOJB04]. *MADSUM* uses a simple auction technique (see 6.4) in which all interested parties submit a basket of bids representing a range of resource consumption and result possibilities. Because the *MADSUM* agents are cooperative and working within the same system, this disclosure of information is not problematic. Wrapper agents that deal with truly external agents (in future implementations) will need to have their own separate protocols for doing business.

There is also work on quickly finding optimal allocations across multiple resources/attributes without having all the bidding agents' preferences disclosed [JR04, PS04](e.g. submitting only one bid in a progressive auction, instead of all possible bids). This is especially interesting since most environments are not collaborative, and such an implementation could expand the domains in which *MADSUM* could operate. However, the failure mechanism that allows *MADSUM* to perform well in a dynamic information environment relies on the availability of secondary bid information, and this would somehow have to be taken into account[1].

Auctions can also be dynamically structured based on a set of formal specifications [LW04]. This allows a system to have a selection of auction types available and then

[1] *MADSUM* agents submit a basket of bids (i.e. a finite subset of their preferences) and so could probably work with such optimal allocation algorithms, but at a cost in speed.

choose the appropriate structure for the conditions at hand, as in [WWW98]. For example, if all agents are internal, collaborative agents, *MADSUM*'s existing style of full disclosure would be appropriate and efficient. But if outside task agents were included, then a partial- or minimal-disclosure auction could be substituted.

Shen et al [SZL04] classify agents on a continuum from "completely self-directed" to "completely externally directed". Self-directed agents do not take into account the ability of other agents to produce utility (whether for themselves or for the system as a whole). Externally directed agents value another agent's gain in utility as their own. This continuum represents how an agent approaches achieving its goals. On a separate continuum, self-interested agents have goals that only reflect their own utility, while co-operative agents have goals that reflect global, or "social" utility. *MADSUM* agents are designed to be a combination of self-directed and cooperative. Their individual goals are the system goals (which in turn are the user's goals), making them cooperative, but the way they go about achieving those goals is to consider only the utility they can derive locally (e.g. by allocating resources to children and assembling the results) which makes them self-directed.

6.2 DECAF

The *MADSUM* decision support architecture uses the services of a separate agent architecture, DECAF, to provide the basic functionality of individual agents in the community. This section outlines features of DECAF and why I chose it for *MADSUM*.

DECAF (Distributed, Environment-Centered Agent Framework) is a toolkit which allows a well-defined software engineering approach to building multi-agent systems. 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 [DS97, SDP+96]: communication, scheduling, execution monitoring, coordination, and eventually learning and

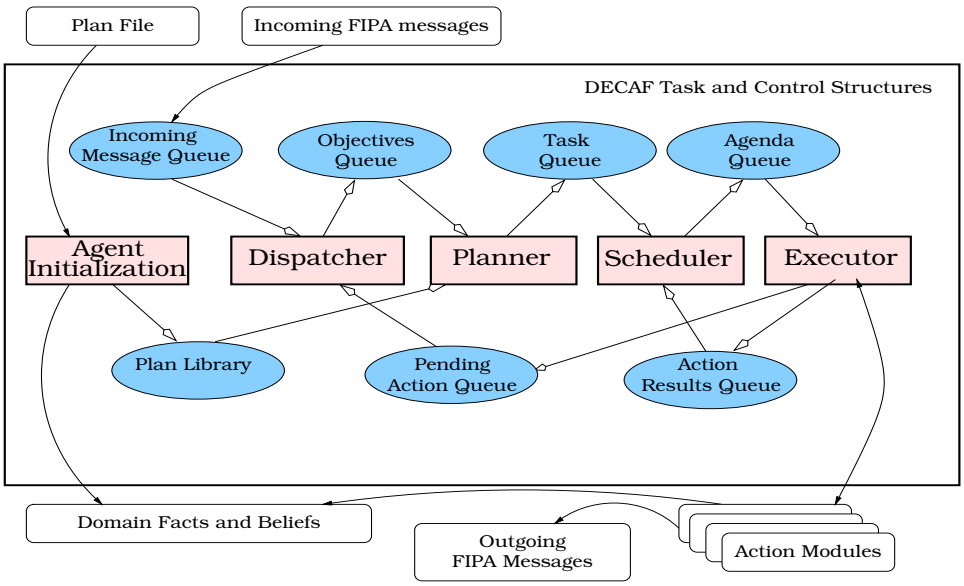


Figure 6.1: DECAF Architecture Overview

self-diagnosis. This is essentially the internal "operating system" of a software agent, to which application programmers have strictly limited access.

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). This paradigm differs from most of the well-known agent toolkits, which instead use the API approach to agent construction (e.g., [Pet96]) where programmers are provided classes and functions with which to design agents. Functionally, DECAF is based on RETSINA [SDP+96] and TÆMS [DL93].

The TÆMS formalism provides a way for agents to explicitly represent trade-offs inherent in scheduling decisions by quantifying a task's expected cost, quality, and duration. This representation allows a task scheduler to make "intelligent" runtime decisions about how to best organize its workload given the computational resources available. For example, TÆMS can represent that the execution of one task in a plan can facilitate the execution of a different one. While *MADSUM*'s text plans include the information necessary to allow the detection of TÆMS-style facilitation, the addition of facilitation links

between plan nodes would fall outside the RST formalism (see Section 2.3).

The control or programming of DECAF agents can be provided via a graphical user interface called the *Plan-Editor*. The Plan-Editor can be used to construct hand-coded hierarchical task networks to be merged into larger, more complete plans, or to view or edit plans. *MADSUM* task agents consist of six task networks created in the Plan-Editor, each containing a number of subtask networks, which in turn eventually decompose into actions to be scheduled by DECAF and performed by the agent.

Figure 6.1 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 incoming FIPA[2] messages).

As shown in Figure 6.1, there are five internal execution modules (square boxes) in the current implementation, and seven associated data structure queues (oval boxes). DECAF is multi-threaded, and thus all modules execute concurrently, and continuously (except for agent initialization).

The Planner monitors the Objectives Queue and plans for new goals, based on the action and task network specifications stored in the Plan Library. A copy of the instantiated plan, in the form of a hierarchical task network corresponding to that goal is placed in the *Task Queue* area, along with a unique identifier and any provisions that were passed to the agent via the incoming message. The Task Queue at any given moment will contain the instantiated plans/task structures (including all actions and subgoals) that should be completed in response to all incoming requests and local maintenance or achievement goals.

---

[2] FIPA is the Foundation for Intelligent Physical Agents, and has written the proposed IEEE standards for agent message protocols.

6.2.1 Why DECAF?

I chose DECAF as *MADSUM*'s underlying agent architecture for four reasons. First, programming agents in DECAF is facilitated by the Plan-Editor. This feature makes the development process easier by allowing agents to be designed by simply assembling graphically designed task structures. Second, I wanted *MADSUM* to take advantage of parallel computation on multiple systems, and DECAF has built-in features to facilitate communication between agents on different systems. Finally, previous work[DWS96] in DECAF has resulted in the development of a Matchmaker agent that allows agents to enter and exit an agent "marketplace." A Matchmaker, as the name implies, allows agents to match their needs with the services provided by other agents (or vice versa). This feature could facilitate the expansion of *MADSUM* into a domain that has existing information source agents, and could also increase the existing implemented system's ability to handle agent failures (by allowing agents to select, at run time, among duplicates of existing source or task agents).

6.3 The *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 (see Figure 6.2), currently consisting of thirteen DECAF software agents. At the lowest level are seven information providers, or "source" agents, that can access information, sometimes from remote sources. Above these sources in the hierarchy are task agents, or "wrappers" that have source agents as children. They are called wrappers because part of their task is to "wrap" the information from the source agents in a package readable by the other agents in the system: in this system, a text plan. At the top of the hierarchy is a Presentation Agent which receives the initial request for decision support from the user and interacts with the task agents (who interact with other task agents and source agents) to produce a message.

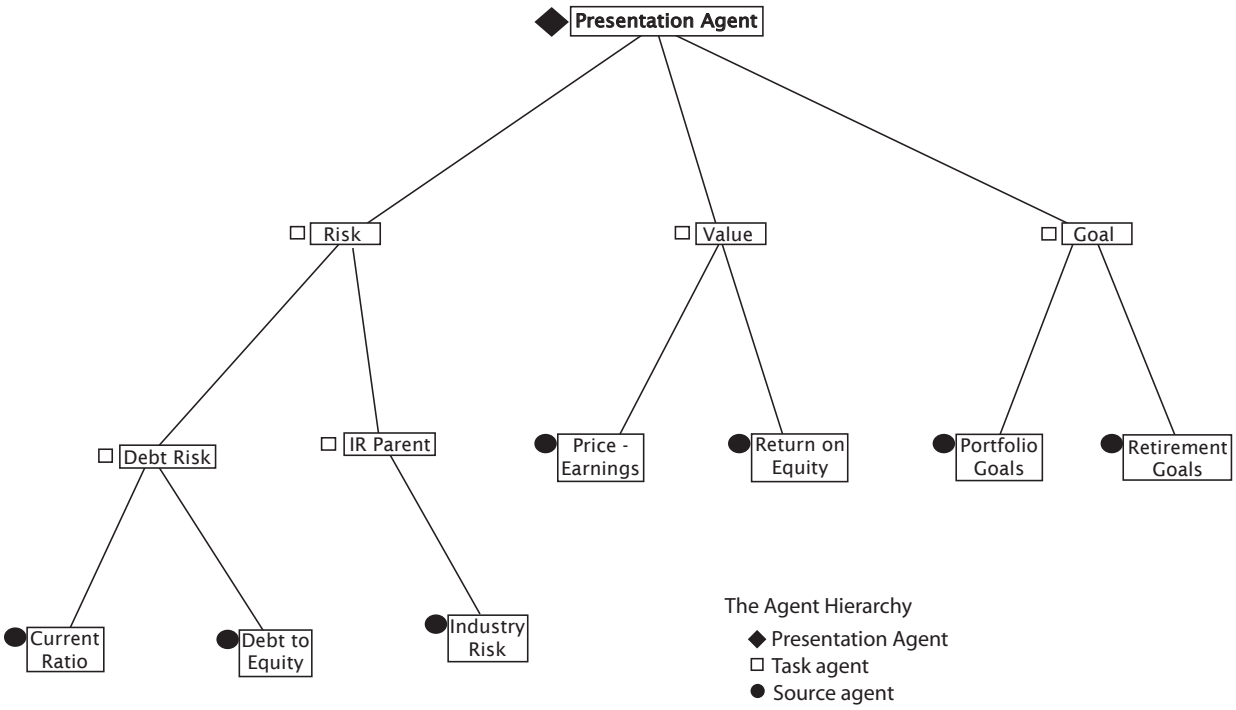


Figure 6.2: The *MADSUM* agent hierarchy

6.3.1 Information source agents

Many of the information providers, or source agents, report the value of a specific financial ratio for a company. A financial ratio[3] is simply a formula involving two or more numbers taken from a company's financial reports. For example, the Current Ratio is found by dividing a company's current assets by current liabilities; these are both numbers found on a financial statement called a balance sheet.

The implemented source agents are:

1. **Current Ratio**: this agent returns the Current Ratio for a given company. The Current Ratio is a financial ratio that indicates the ability of a company to cover short-term obligations.

[3] The financial ratios used here are widely used and described. For a thorough analysis of the import of each ratio used here (and many others) see [Bra02].

2. **Debt-to-Equity**: a financial ratio agent. This ratio is an indication of how highly leveraged a given company is (i.e. how much debt they have relative to the value of the company).

3. **Industry Risk**[4]: an agent that returns a broad assessment of the degree of risk involved in the company's primary industry, where risk here is taken to mean volatility as reported by Barron's [Bar].

4. **Price-Earnings**: a financial ratio agent. The Price/Earnings ratio gives an indication of how expensive a stock is relative to its recent earnings performance.

5. **Return-on-Equity**: a financial ratio agent. This ratio is an indication of the yield an investor would have derived in a recent period. Note that the yield (return) does not consider capital appreciation.

6. **Portfolio Goal**: an agent that considers a potential investment's impact on the user's portfolio allocation goals. In particular, the user model includes the user's intended balance between equities (stocks, or ownership in another organization) and securities (bonds, i.e. debt of another organization). The agent reports on whether the investment under consideration moves the user closer to or further from their stated goals, and may also note if a purchase would concentrate the user's assets too highly in a single investment.

7. **Retirement Goal**: an agent that considers an investment and the user's portfolio in light of the user's number of years to retirement. The agent compares a percentage of the user's annual consumption to the user's cash equivalent (securities) balance. This is motivated by the philosophy that as retirement approaches, investments that

[4] This agent was not part of the original set of financial source agents, but was added to *MADSUM* later as an exercise to test the ease of including a new agent and its information. Adding the agent and its wrapper (see IR Parent) took about four hours from starting the PlanEditor to running successful test data.

will be consumed within five years should be moved out of the more volatile equities and into securities. The agent comments if insufficient securities are available to cover near-future retirement needs. Note that there are no near-future retirement needs if the user is more than five years from retirement.

The agents above that report financial ratios have a two tier price system. For the low price they simply report a ratio. This is useful, since many ratios have values with meanings that hold across companies and even industries. For example, a Current Ratio over two is considered a sign of (short-term) strength in any industry. For a higher price the agent will report on this company's ratio relative to that of the company's primary industry classification (e.g. DCX (Daimler-Chrysler's) industry classification is auto manufacturing). This is more valuable information since it takes into account information specific to an industry. For example, firms in the contracting business are expected to have relatively high debt loads, so their Debt-to-Equity ratios will look high in an absolute sense, but not when compared to other contractors.

6.3.2 Internal task agents

Task agents are internal to the hierarchy, i.e. they are neither leaves (those would be source agents) or at the root (the top, which is the Presentation Agent). Task agents may serve as "wrappers", using domain knowledge to convert the information from their children that are source agents into simple text plan trees via templates. Wrappers are a research topic on their own. Of particular relevance is work in automatically generating wrappers for simple web sources like the financial ratios described here [CHJ02, Kus00, AK97, KWD97, Sod97]. (The existence of these automatic and semi-automatic wrapper generators is a further argument for the development of systems such as *MADSUM*, that can integrate the results provided by multiple, wrapped sources.) Task agents also negotiate about the contents of messages, and if they have multiple children they integrate the text plan trees of their children into a single plan tree.

The three highest-level task agents are RISK, VALUE, and GOAL, and their tasks correspond with three common ways to analyze the suitability of an investment for a particular investor. Risk analysis examines how volatile an investment may be, based on certain relevant financial characteristics. Similarly, value analysis looks at attributes of a company's financial record that may give insight into differences between an investment's market value and its price. Finally, users make investments to achieve goals, and thus investments can be evaluated in terms of their contributions to different user goals (e.g. reaching certain portfolio benchmarks). These three task agents communicate with the Presentation Agent above (see Figure 6.2), and either with task agents below (further dividing topics into specialties, e.g. risk is divided into debt risk and industry risk) or with source agents below (e.g. GOAL communicates directly with the source agents PORTFOLIO GOAL and RETIREMENT GOAL). For simplicity of design, task agents have as children either other task agents or source agents, but not both. The task agents are as follows:

1. **Risk**: This task agent is responsible for messages about risk information. The task is further divided into subtasks for task agents DEBT RISK and INDUSTRY RISK.

2. **Debt Risk**: This agent handles source agents that report information on debt-related aspects of a company's financial picture. The source agents that fit this description are Current Ratio and Debt-to-Equity, so this agent is parent to both. This agent's parent is the task agent RISK.

3. **IR Parent**: This agent is solely a wrapper for Industry Risk. Since task agents communicate either with other task agents or with source agents, but not both, INDUSTRY RISK cannot communicate directly with RISK, thus requiring this intermediary task agent. All source agents need wrappers to convert their data to text plans, but because there are no other source agents reporting this type of information, INDUSTRY RISK is an only child (see footnote for INDUSTRY RISK on page

130). This task agent's parent is the task agent RISK.

4. **Goal**: parent to Portfolio Goal and Retirement Goal, this agent wraps sources that consider the user's explicit or implied financial goals. The parent of GOAL is the Presentation Agent.

5. **Value**: parent to Price-Earnings and Return-on-Equity, this agent wraps sources that provide information that can be useful in evaluating a company's recent performance relative to its share price. The parent of VALUE is the Presentation Agent.

The internal task agents of the hierarchy each have the capacity to make independent decisions about which information to acquire from their children, and how to recover when information from agents below fails to meet expectations (see Section 6.6). This distributed structure facilitates 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[Jen95b]. See Section 6.7.

The final agent is the **Presentation Agent**: parent to RISK, VALUE, and GOAL, this agent receives the initial task message from the user via the GUI interface, and responds with the text realized from the final text plan tree. Otherwise, it behaves like the other task agents, except that it has no parent. This agent has no domain specific knowledge, since it does not wrap any sources.

6.4 The Negotiation Process

The negotiation process consists of four parts based on the common FIPA Contract Net Interaction Protocol[fIPA02, OVDPB01] (see Figure 6.3). In the simplest version of the protocol, an agent A requests that agent B make a proposal (bid) to execute a task that has certain specifications. B either rejects the invitation or responds with a proposal. In the third stage, A either accepts or rejects B's proposal, and if A accepts, the fourth stage

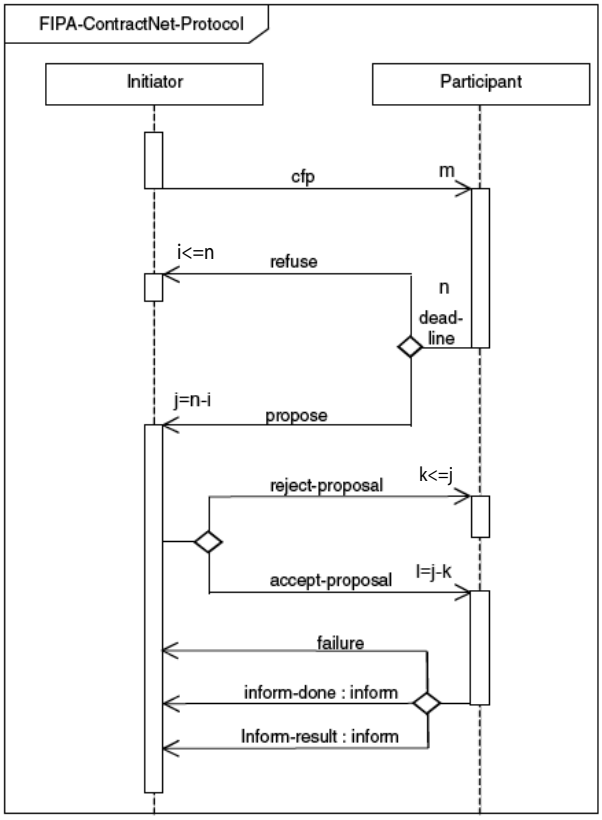


Figure 6.3: The FIPA Contract Net Interaction Protocol Specification[fIPA02, OVDPB01]. A generic, standardized recipe of messages and responses that allows agents to contract with other agents for services/goods, with time starting at the top of the diagram (call for proposal) and failure or result reported at the bottom.

is B fulfilling the proposal, and returning results to A, if any. This protocol provides the outline for the behavior of agents in *MADSUM*.

The form that the protocol takes in *MADSUM* is as follows:

1. A decision-support task (e.g. consider a stock purchase being considered by the user), a utility function, and a set of soft and hard constraints are passed to the agent at the top of the *MADSUM* hierarchy, the Presentation Agent.

2. The Presentation Agent then forwards this information to its children (the high level

task agents RISK, VALUE, and GOAL) as a solicitation for bids, and the solicitation is propagated down the agent tree to the source agents.

3. Starting at the source agents (leaves), agents bid by submitting multiple estimates for results they expect to be able to provide, expressed as a series of utility attribute/value pairs.

4. Agents propagate bids back up the tree; at each level, agents select among the bids from their child agents, and then submit a single bid to their parent agent.

5. Starting with the Presentation agent, agents commit to specific bids from their children; and finally

6. 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. The next three sections will describe the protocol in more detail.

6.4.1 Passing Request Information Down the Hierarchy

First, the top-level Presentation Agent receives a request from the user (via the graphical user interface) to provide information that will help in a decision about a proposed investment; this triggers a bidding process. Each agent in the hierarchy distributes the request, along with hard constraints and the utility function, to its children. The request is seen by each agent as a solicitation for "bids". The request progresses down to the information agents at the leaves of the agent hierarchy. To estimate the potential significance (Decision Specificity or DS, see 5.4.4) to the user of the information that they might provide, the lowest level agents use knowledge of the domain 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. Agents will provide multiple bids, representing different trade-offs of overall utility provided and resource consumption.

An important part of the *MADSUM* architecture/philosophy is implicit in this first stage. Note that the agents in the hierarchy do not allocate separate quantities of information or resources when passing the information to agents below them. Instead, all agents know the full amount of resources available, and the original request for information. While this practice presents challenges (see Section 6.6), it also makes it possible for a low level agent to consider every means to contribute utility to the user. In particular, a low-level agent can consider what utility it could provide if it consumed all of the allotted resources itself. Under some circumstances, this would allow the agent to consider, for example, purchasing high cost information that would be of very high value to the user.

6.4.2 Submission of bids by agents

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 terms of the attribute values (thus providing options during later parts of the task). At each level i, parent agents will consider various combinations of the bids submitted by children in level $i + 1$. The parent agents evaluate the bids in terms of utility, select a "basket" of combinations, and propagate new bids representing the basket up the tree. This occurs recursively to the top of the tree. Though agents submit multiple bids, each combination includes at most one contribution from each child agent. Since each bid is a fully formed text plan designed in response to the original request, including more than one full response would probably result in duplicate information. Thus the complexity of the problem may be constrained by limiting the number of children of an agent. More precisely, given a maximum number of m bids per child, and including the possibility that the child will not have any bids included in the basket, then there are $m + 1$ ways for a child to submit bids. For n children, and excluding the combination which includes no

bids, the number of combinations considered by a task agent has an upper bound of:

$$(m+1)^n - 1 \tag{6.1}$$

Though the hierarchical design of *MADSUM* can limit the combinatorial auction complexity by restricting the number of children a given agent has, there is also work towards making the general case of such auctions more tractable by imposing some minor restrictions on the process; see [WW00, CS04].

For any auction more complex than a single item at a monetary (or single resource) price, the process of mapping agent bids into resource allocations is problematic. The bid process in *MADSUM*, where agents suggest a probable utility payoff in exchange for allocation of multiple resources, is more complex, especially when aspects of the dynamic environment (incomplete information, possible failures) are considered. The agents participating in each task agent's "auction" constitute a "market game":

> Market games corresponding to even moderately complex scenarios are notoriously difficult to solve. That is, except for the simplest market mechanisms (e.g., a oneshot auction for a single item, or a mechanism specially designed to have dominant strategies), deriving a Bayes-Nash equilibrium is not analytically tractable.[WMMRS03]

The *MADSUM* process for allocating resources is not a standard auction process, in which a single bidder will bid against other agents over time and eventually either receive all or none of the resources from the controlling agent (in *MADSUM*, a parent). First, while all agents submit multiple bids, they submit the bids simultaneously, rather than over time. Second, the parent agent controlling the auction is not considering which single bid will afford the most utility within the available resources, but rather is considering combinations of bids (and the combinations considered will include combinations of one, i.e. single bids.) Considering combinations changes the complexity of the process to that of a combinatorial auction based on complementary resources. *Complementary* in this sense means that the value of one resource is to some degree dependent on the availability of other resources[WMMRS03]. While this is taken into account to some degree

by the utility function which takes multiple resources/attributes into consideration, the *MADSUM* allocation process does not consider more complex complementary relations, such as the *facilitates*[5] relation in TÆMS[DL93].

Note that as the requests being passed down the hierarchy inform each agent of the total resources available, it is possible for a single agent to submit a bid that would consume all available resources. Similarly, it is possible for any combination of agents to submit bids using resources in such a way that other agents are excluded. Both of these cases must be possible so that one or more agents can offer the highest utility options available. Consider an example where agent A has a very high cost, high utility result available. If all other agents have low utility results, then the best solution may be for the system to devote all resources to agent A. However, if agents can only submit one bid (or a set of similar high-utility bids), then each agent above A will only submit A's bid, and it won't be possible for agents further up in the hierarchy to consider combinations of bids from other agents that may outperform A's bid. The system's fundamental strategy for addressing this problem is the presence of alternative bids. Since alternatives are generated and propagated at each level, parents will always be presented with a range of widely varying bids when they are available.

6.4.3 Alternative bids

Alternative bids are chosen in such a way that they serve two functions in *MADSUM*. First, when included in a set of bids submitted by an agent to its parent, they reduce the likelihood that all the bids will look similar in their utility and their consumption of resources (as they might if the n highest utility bids were submitted). This means that the parent will have a broader range of choices with which to make basket combinations. Second, the alternative bids provide a "back up" for the failure mechanism (see Section 6.6).

[5] The *facilitates* relation takes into account that some agent A's actions increase the utility of the actions of a different agent B, even though they may not be required by B.

Should the agent's first bid not perform as expected, the failure mechanism will examine other bids submitted by the agent; the alternative bids are designed to provide flexibility in the face of prior failures, as described below.

Alternative bids are bids that are chosen based on how different they are from the high utility bid and from one another. In particular, they are not chosen based on utility, but rather on how dissimilar the attribute values in the bids are from one another. The purpose of alternative bids is to ensure some degree of diversity in the bid basket presented to a parent. 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 unattractive. If all the alternatives were chosen to maximize utility, then *all* might well be unattractive. Bid diversity will also likely be valuable under certain failure circumstances (e.g. a network failure), but that is not demonstrated in this thesis.

Selecting alternative bids thus implies the ability to select bids dissimilar from the high-utility bids. One possible approach would be to simply select low utility bids. This faulty strategy assumes that if another utility function is used, it will be some kind of inverse of the original utility function. Just because low cost and high DS are desirable now does not imply that high cost and low DS will be attractive in dynamic circumstances.

Another possible approach would be to enumerate the terms of the utility function and determine *a priori* which ones are most likely to change, and in what direction. This approach assumes a great deal of predictability in the dynamic environment, which is counter to the *MADSUM* assumption of an unpredictable environment. Also, *MADSUM* is designed to take an arbitrary set of utility attributes, and even if such predictions were plausible, they would have to constantly be updated as attributes were added.

My approach is to consider the bids themselves, absent any consideration of utility. Bids are grouped by similarity, and the alternative bid selection tries to ensure that each different group of bids has at least one representative in the basket of bids passed up

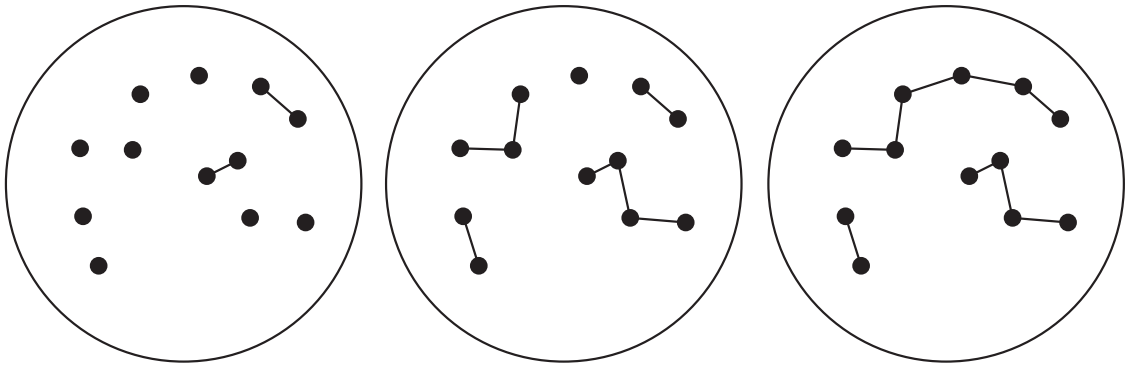


Figure 6.4: An example showing three stages of Kruskal's algorithm. The leftmost stage has had two edges added. The shortest edge is added to the graph at each step, growing a forest of minimum spanning trees. *MADSUM* stops the process when enough edges have been added to reach the desired number of trees (here three).

---

to the parent. (This approach contrasts with the decision-theoretic generation systems FLIGHTS[MFLW04] and MATCH[WWS$^+$04, JBV$^+$]. In these systems, a user cannot change his preferences about an attribute after setting them.)

To determine the degree of similarity between bids, the numeric values in a bid (e.g. length, cost, DS) are treated as a vector projected onto a unit sphere, and then the angles between the vectors are used to estimate difference as distance along the sphere's surface[Sal88]. Based on this measurement, Kruskal's algorithm[CLR92] is used to grow a forest of minimum spanning trees, gradually consolidating adjacent (similar) trees by connecting the two trees with the shortest distance between them until the desired number of trees is reached. (The system default is three alternatives, and would stop the process at four trees. The extra one is for the tree that will contain the high utility bid.)(see Figure 6.4). At this point, the intent is that all bids that are most similar to the high bid will be in the same tree as the high bid; each other tree will have some set of closely

$$Utility = 5*EndPlateauLinear(DS_r, 10)+5*EndPlateauLinear(DS_v, 10)+5*Normal(length, 70)$$

1. Bids from child A:
   $length_{expected} = 40, DSr_{expected} = [[3\ 5\ 2][0\ 0\ 0]], DSv_{expected} = [[0\ 0\ 0][0\ 0\ 0]]$
   $length_{expected} = 25, DSr_{expected} = [[2\ 2\ 1][0\ 0\ 0]], DSv_{expected} = [[0\ 0\ 0][0\ 0\ 0]]$
   $length_{expected} = 15, DSr_{expected} = [[3\ 3\ 1][0\ 0\ 0]], DSv_{expected} = [[0\ 0\ 0][0\ 0\ 0]]$

   Bids from child B:
   $length_{expected} = 40, DSr_{expected} = [[0\ 0\ 0][0\ 0\ 0]], DSv_{expected} = [[3\ 5\ 2][0\ 0\ 0]]$
   $length_{expected} = 25, DSr_{expected} = [[0\ 0\ 0][0\ 0\ 0]], DSv_{expected} = [[2\ 2\ 1][0\ 0\ 0]]$
   $length_{expected} = 15, DSr_{expected} = [[0\ 0\ 0][0\ 0\ 0]], DSv_{expected} = [[3\ 3\ 1][0\ 0\ 0]]$

2. Six highest utility bids representing combinations of children of A and B:
   $length_{expected} = 80, DSv_{expected} = [[3\ 5\ 2][0\ 0\ 0]], DSr_{expected} = [[3\ 5\ 2][0\ 0\ 0]]\ U = 59.1$
   $length_{expected} = 65, DSv_{expected} = [[2\ 2\ 1][0\ 0\ 0]], DSr_{expected} = [[3\ 5\ 2][0\ 0\ 0]]\ U = 54.6$
   $length_{expected} = 65, DSv_{expected} = [[3\ 5\ 2][0\ 0\ 0]], DSr_{expected} = [[2\ 2\ 1][0\ 0\ 0]]\ U = 54.6$
   $length_{expected} = 55, DSv_{expected} = [[3\ 5\ 2][0\ 0\ 0]], DSr_{expected} = [[3\ 3\ 1][0\ 0\ 0]]\ U = 45.4$
   $length_{expected} = 55, DSv_{expected} = [[3\ 3\ 1][0\ 0\ 0]], DSr_{expected} = [[3\ 5\ 2][0\ 0\ 0]]\ U = 45.4$
   $length_{expected} = 50, DSv_{expected} = [[2\ 2\ 1][0\ 0\ 0]], DSr_{expected} = [[2\ 2\ 1][0\ 0\ 0]]\ U = 25.3$

3. Three dissimilar alternative Bids representing combinations of children of A and B:
   $length_{expected} = 30, DSr_{expected} = [[3\ 3\ 1][0\ 0\ 0]], DSv_{expected} = [[3\ 3\ 1][0\ 0\ 0]]\ U = 20.5$
   $length_{expected} = 15, DSr_{expected} = [[0\ 0\ 0][0\ 0\ 0]], DSv_{expected} = [[3\ 3\ 1][0\ 0\ 0]]\ U = 10.0$
   $length_{expected} = 15, DSr_{expected} = [[3\ 3\ 1][0\ 0\ 0]], DSv_{expected} = [[0\ 0\ 0][0\ 0\ 0]]\ U = 10.0$

Figure 6.5: Bids and dissimilar alternatives and their utility under the given function.

related bids that are similar to one another, and dissimilar from the high bid (and its tree). Alternative bids can then be chosen from the trees not already represented in the set of high utility bids. Currently the highest utility bid from each tree is used.

The cost of using Kruskal's algorithm for n bids is $O(n^2log(n))$, derived as follows. Since Kruskal's computes the distance between every pair of bids, for n bids it computes $n(n-1)/2$ distances (one distance, or edge, between each possible pair of bids) which is order $O(n^2)$. Then the pairs must be sorted so that they can be added to the existing trees in increasing order of distance; for n^2 distances the sort costs $O(n^2log(n^2))$, which simplifies to $O(2 * n^2log(n))$, or $O(n^2log(n))$. The sorted list is traversed at a cost

$length_{expected} = 80,\ DSv_{expected} = [[3\ 5\ 2][0\ 0\ 0]],\ DSr_{expected} = [[3\ 5\ 2][0\ 0\ 0]],\ U = 59.1$
$length_{expected} = 65,\ DSv_{expected} = [[2\ 2\ 1][0\ 0\ 0]],\ DSr_{expected} = [[3\ 5\ 2][0\ 0\ 0]],\ U = 54.6$
$length_{expected} = 65,\ DSv_{expected} = [[3\ 5\ 2][0\ 0\ 0]],\ DSr_{expected} = [[2\ 2\ 1][0\ 0\ 0]],\ U = 54.6$
$length_{expected} = 55,\ DSv_{expected} = [[3\ 5\ 2][0\ 0\ 0]],\ DSr_{expected} = [[3\ 3\ 1][0\ 0\ 0]],\ U = 45.4$
$length_{expected} = 55,\ DSv_{expected} = [[3\ 3\ 1][0\ 0\ 0]],\ DSr_{expected} = [[3\ 5\ 2][0\ 0\ 0]],\ U = 45.4$
$length_{expected} = 50,\ DSv_{expected} = [[2\ 2\ 1][0\ 0\ 0]],\ DSr_{expected} = [[2\ 2\ 1][0\ 0\ 0]],\ U = 25.3$
$length_{expected} = 30,\ DSv_{expected} = [[3\ 3\ 1][0\ 0\ 0]],\ DSr_{expected} = [[3\ 3\ 1][0\ 0\ 0]],\ U = 20.5$
$length_{expected} = 40,\ DSv_{expected} = [[2\ 2\ 1][0\ 0\ 0]],\ DSr_{expected} = [[3\ 3\ 1][0\ 0\ 0]],\ U = 20.0$
$length_{expected} = 40,\ DSv_{expected} = [[0\ 0\ 0][0\ 0\ 0]],\ DSr_{expected} = [[3\ 5\ 2][0\ 0\ 0]],\ U = 20.0$
$length_{expected} = 40,\ DSv_{expected} = [[3\ 3\ 1][0\ 0\ 0]],\ DSr_{expected} = [[2\ 2\ 1][0\ 0\ 0]],\ U = 20.0$
$length_{expected} = 40,\ DSv_{expected} = [[3\ 5\ 2][0\ 0\ 0]],\ DSr_{expected} = [[0\ 0\ 0][0\ 0\ 0]],\ U = 20.0$
$length_{expected} = 15,\ DSv_{expected} = [[3\ 3\ 1][0\ 0\ 0]],\ DSr_{expected} = [[0\ 0\ 0][0\ 0\ 0]],\ U = 10.0$
$length_{expected} = 15,\ DSv_{expected} = [[0\ 0\ 0][0\ 0\ 0]],\ DSr_{expected} = [[3\ 3\ 1][0\ 0\ 0]],\ U = 10.0$
$length_{expected} = 25,\ DSv_{expected} = [[2\ 2\ 1][0\ 0\ 0]],\ DSr_{expected} = [[0\ 0\ 0][0\ 0\ 0]],\ U = 6.8$
$length_{expected} = 25,\ DSv_{expected} = [[0\ 0\ 0][0\ 0\ 0]],\ DSr_{expected} = [[2\ 2\ 1][0\ 0\ 0]],\ U = 6.8$

Figure 6.6: Bids resulting from the fifteen possible combinations of child bids, ordered by utility.

of $O(n^2)$, adding the pairs of bids to minimum spanning trees. Thus the dominant term in the cost is the sort, or $O(n^2 log(n))$. To prevent the size and associated computation time from getting out of hand, if the number of bid combinations being considered exceeds one hundred, *MADSUM* selects a random sample of 100 bid combinations from which to choose the alternatives. This keeps the alternative selection process under one second per agent as implemented.

Consider a small example where an agent P has two children, A and B. Each child agent submits three bids to their parent P: two bids representing single messages from sources and one bid representing the combination of the first two bids. Thus P receives a total of six bids from two sources (see Figure 6.5, part 1).

P must now determine what bids to send to her parent. P considers all possible combinations, in this case fifteen combinations with either zero or one bid from each child:

$$(A_1)\ (A_2)\ (A_3)\ (A_1, B_1)\ (A_2, B_1)\ (A_3, B_1)\ (A_1, B_2)\ (A_2, B_2)$$

New utility function:
$Utility = 5{*}EndPlateauLinear(DS_r, 10){+}5{*}EndPlateauLinear(DS_v, 10){+}5{*}Normal(length, 35)$

1. Six previous highest utility bids under new utility function
   $DSv_{expected} = [[3\ 5\ 2][0\ 0\ 0]], length_{expected} = 80, DSr_{expected} = [[3\ 5\ 2][0\ 0\ 0]]$ **U=33.3**
   $DSv_{expected} = [[3\ 5\ 2][0\ 0\ 0]], length_{expected} = 65, DSr_{expected} = [[2\ 2\ 1][0\ 0\ 0]]\ U = 23.3$
   $DSv_{expected} = [[2\ 2\ 1][0\ 0\ 0]], length_{expected} = 65, DSr_{expected} = [[3\ 5\ 2][0\ 0\ 0]]\ U = 23.3$
   $DSv_{expected} = [[3\ 5\ 2][0\ 0\ 0]], length_{expected} = 55, DSr_{expected} = [[3\ 3\ 1][0\ 0\ 0]]\ U = 27.2$
   $DSv_{expected} = [[3\ 3\ 1][0\ 0\ 0]], length_{expected} = 55, DSr_{expected} = [[3\ 5\ 2][0\ 0\ 0]]\ U = 27.2$
   $DSv_{expected} = [[2\ 2\ 1][0\ 0\ 0]], length_{expected} = 50, DSr_{expected} = [[2\ 2\ 1][0\ 0\ 0]]\ U = 16.6$
2. Three previously selected alternative Bids under new utility function:
   $length_{expected} = 30, DSr_{expected} = [[3\ 3\ 1][0\ 0\ 0]], DSv_{expected} = [[3\ 3\ 1][0\ 0\ 0]]$ **U=45.8**
   $length_{expected} = 15, DSr_{expected} = [[0\ 0\ 0][0\ 0\ 0]], DSv_{expected} = [[3\ 3\ 1][0\ 0\ 0]]\ U = 10.5$
   $length_{expected} = 15, DSr_{expected} = [[3\ 3\ 1][0\ 0\ 0]], DSv_{expected} = [[0\ 0\ 0][0\ 0\ 0]]\ U = 10.5$

Figure 6.7: Three responses, derived from different soft constraints and priority settings.

$$(A_3, B_2)\ (A_1, B_3)\ (A_2, B_3)\ (A_3, B_3)\ (B_1)\ (B_2)\ (B_3)$$

These combinations are converted to single bids for comparison purposes; the fifteen resulting bids are shown in Figure 6.6. If P chooses to submit the six highest utility bids, then the six are as shown in Figure 6.5, part 2.

Now consider what happens if during the decision support process the user cuts the desired length from 70 to 35. If the six highest bids were forwarded to the parent, then the previous high-utility bid of length 80 remains the high-utility bid of those six. But none of the old bids is now particularly desirable, since all of their utilities have dropped substantially as shown in Figure 6.7, part 1. Furthermore, if the change in length had been a hard constraint change instead of a preference, all six would have failed.

In contrast, if P submitted only the three highest utility bids and three dissimilar alternates[6] (see Figure 6.5, part 3) chosen using the algorithm described, the highest utility bid would still be available, with two potential backups, to maximize utility under the

[6] These are the system defaults.

original function. In addition, under the new utility function one of the alternate bids actually becomes the high utility bid (see Figure 6.7, part 2). (In this case it was the highest of all possible bids, but the point here is merely that alternatives can become preferable under alternate utility functions, even if they are not optimal.)

Of course, in practice it is not possible to know *a priori* how many alternative bids, and hence tree groups, one may need. The presence of any n alternative bids is no guarantee that they will be a better fit for the environment at some time t. Still, having some k highest utility bids and n alternative bids is demonstrably better in some circumstances than having the $k + n$ highest utility bids. Also, note that while in a stable environment $k = 1$ would be sufficient for determining the highest possible utility bid from a particular agent, (since the utility function would not change, alternatives would not be helpful), nevertheless alternative bids are needed to provide choice to this agent's parent during the negotiation process that allocates resources. (Note that the presence of alternatives does nothing to ensure that the bid eventually selected will prove usable further up the decision hierarchy; alternatives reduce the performance penalty associated with such events, as explained in Section 6.6).

Finally, note that the expected utility of an alternative bid set, under the original utility function, must be less than or equal to the lowest of the high utility bid sets. To show this, suppose that there are three high utility bid sets of equal or decreasing utility B_{h1}, B_{h2}, B_{h3}, and three alternate utility bid sets of equal or decreasing utility B_{a1}, B_{a2}, B_{a3}. All utilities are measured under some original utility function U_1. Since all bid sets are considered together when choosing the highest utility bid sets, then no alternative bid set B_{a_i} can have a utility higher than B_{h3}; otherwise B_{a_i} would be one of the high utility bid sets, not an alternate. It is conceivable that all the bid sets could have the same utility under U_1.

6.4.4 Allocating Resources under Hard Constraints

Users are permitted to set constraints on the consumption of resources such as length, cost, or time. These constraints are considered "hard", i.e. agents' results must conform to these constraints or be considered failures. (This is in contrast to the "soft" constraints that reflect aspects of user preferences in the utility function; when a result exceeds a soft constraint there is not a failure, merely degraded utility.) *MADSUM*'s preferred mode of operation is to avoid such constraints, since they place restrictions on the ability of agents to respond to the dynamic environment, as described in Section 6.4.5.

If the user does specify a hard constraint for a resource (such as an absolute limit of 100 words on length), then the system separately allocates the length resource across the agents whose bids were accepted. Agents are not permitted to exceed an allocation limited by a hard constraint. This avoids the possibility of peer agents in one level of the hierarchy each (separately) deciding to exceed the resource use estimate specified in their original bid, thus potentially exceeding the hard constraint when the peer agents' text plans are joined. In the case of length, this problem could be solved (at some non-zero cost in time and/or money) via the failure protocol, but if the resource in question were cost, then by the time the failure is detected, agents further down in the hierarchy would have spent the money in question, and the failure protocol would not be able to reverse the expenditure.

The complement of this problem is the reason that users should avoid specifying hard constraints: when agents fail to use as much of a resource as they expected, there is no simple way in a distributed system to make the unused resources available to other agents that might use them to advantage.

To allocate a resource across agents when the resource is bounded by a hard constraint, the resource is first distributed according to the accepted set of bids. (Since the set of bids was accepted, the set fit within constraints, and there is at least a sufficient amount of the resource to cover all the accepted bids.) Any excess of a constrained resource is

distributed according to the portion of the overall utility that an agent contributes, so that the largest extra supply of a resource goes to the agent supplying the biggest portion of utility, not to the agent using most of the resource.

6.4.5 Allocating Resources via Utility Envelopes

The topmost agent (the Presentation Agent) selects the highest utility bid and propagates it back down the tree as a commitment of resources and an associated expectation of utility (in a utility envelope) that will be produced. This section describes why this method of allocating resources is an effective approach to the problems created by *MADSUM*'s dynamic environment.

MADSUM assumes that it is difficult to predict the utility of an information-gathering task before the task is performed. This is a result of the dynamic environment in which *MADSUM* operates, since not only does information change rapidly, but the user can also modify the utility function at any time. Changing information results in changes in the value of the information content (and thus its utility to the user), but also in the message attributes of the information. A changing utility function directly affects the utility calculation, and may be caused by changes in attribute terms such as new user preferences regarding cost, time, or message length. Yet if resources such as cost, time, and length are limited, an attempt must be made to distribute those resources across agents in a way that maximizes utility.

Under certain circumstances, it would be possible for an agent to determine the degree to which each attribute of a result contributes to overall utility and allocate resources in a decision-theoretic manner. First, the utility function would have to be static (which is not the case for *MADSUM*), so that allocation of individual resources could be reasonably expected to deliver a certain utility. Second, attribute values would have to be additive. Because the user chooses a separate function to be applied to each attribute term of the utility function, it doesn't make sense to talk about an individual result's contribution to utility. For example, suppose the user selects the function $StartPlateauNorm$

(see Section 5.4.3) for the LENGTH attribute such that a length of up to 50 provides a utility of 1, but a length beyond 90 provides near zero utility. Now suppose two agents, C1 and C2 expect to return texts of length 50. There is no straight-forward method for the parent P to discover that if either or both agents reduce length, overall utility will probably rise; instead, such a discovery would require sophisticated artificial intelligence. Similarly, 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.

Since the system cannot determine an individual result's contribution to utility, or the contribution of one attribute across a set of results, it does not make sense to allocate specific resources to those individual results. Thus agents submit not one, but a variety of bids, and the parent can then explore combinations of bids without needing to "understand" why one combination provides higher utility than another.

MADSUM's approach is to allocate expected utility instead of resources (except in the case where a hard constraint has been placed on a resource, which is discussed in Section 6.4.4). This 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. In *MADSUM*, the objective is the utility that an agent is expected to deliver, given certain resources.

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. A utility envelope is a number that tells the agent the

lower limit of acceptable variation from the expected utility[7]. The purpose of an envelope is to allow some flexibility in accepting responses that are not exactly as negotiated. This strategy reflects an agent's desire to avoid the real costs, in agent time, computation, and communication, associated with rejecting a response and trying to find an alternative.

If an agent has bid an alternative in addition to its accepted bid, then the total cost to the system of rejecting the result of the accepted bid is reduced, since finding that alternative is trivial. Thus when an agent has an alternative bid, the alternative bid's utility becomes the minimum acceptable utility. The utility envelope is the difference between the original bid's expected utility and the next best alternative. Thus any variation in the result utility returned by the agent is acceptable if it meets or exceeds the utility expected from the next best alternative. Even if the next best alternative has the same utility expected from the original bid, there is a cost associated with abandoning the original and communicating with the agent who submitted the alternative. Therefore a system default minimum envelope percentage can be set so that the original bid will not be considered a failure unless its result drops below the minimum envelope percentage or the utility expected from the next best alternative, whichever is higher.

To clarify the envelope concept, consider a case where the only attribute of the result that is changing is the cost, i.e. 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 higher price. Under what circumstances would this deter you from the purchase? If the cost was very close, e.g. $10.02, you might decide that purchasing the widget at a higher price was still your best course of action. However, if the widget was now $10.80 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

[7] It is called the "envelope" rather than "lower limit" since the system has the ability to represent both high and low limits for utility and utility attributes, resulting in an acceptable range, or envelope for each one. Only the lower limit for utility is currently used.

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). In this example, the original expected utility would have been based on a cost of $10.00, and the utility envelope would have reflected the (lower) utility of the next best alternative (cost $10.20) or a default percentage of the original utility (whichever was higher).

The exception to calculating the envelope based on the next best alternative occurs when there is no alternative, or when the alternative has such low utility that the agent would prefer to report failure and have alternatives investigated at a higher level. In *MADSUM* the trigger level for reporting failure is currently 95 percent, i.e. if no alternative bid has more than 95 percent of the original expected utility, then the envelope is set at 95 percent. The default utility envelope setting of 95 percent was set arbitrarily for our testing purposes to allow flexibility in the presence of minor changes of attributes, thus preventing failure processing unless actual utility was below expected utility by more than five percent. Clearly the setting could depend on the domain, total resource cost and availability, or user preferences, and might even need to vary across attributes. Although this coarse implementation of the default envelope concept could be improved upon, its inclusion in the system is noteworthy as a part of the overall effort to address possible failures.

One advantage to allocating utility 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 expected contributions. 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, *MADSUM* consists of cooperative agents which share a common (user-provided) 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 for addressing results that do not meet specifications (see Section 6.6).

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 text that is 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 occurring again, and being repeated at each level of the tree [YH95]. A system can try to prevent combinations of agents from over-allocating resources in response to bids (which would require a combinatorial number of communications among agents) or the system can attempt to manage such situations when they occur. The *MADSUM* architecture addresses this issue by allowing an agent to make sub-optimal allocations which result in the agent's failure to deliver expected utility, and then focusing on recovering from the resulting failures gracefully (see Section 6.6).

6.4.6 Summary of the *MADSUM* negotiation process and its benefits

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 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 attributes, 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 an agent's failure to deliver an expected result.

6.5 Execution: gathering, integrating and propagating results

When the commitment (utility envelope) has travelled down the hierarchy and 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 information agents examine or query an external source (currently a data file, but potentially a website or database) and retrieve information.

The raw information is passed from the lowest-level information source agents at the leaves of the hierarchy to their parent task agents (wrappers), which examine the information using domain specific expertise (e.g. a Current Ratio over two is a good thing) and map the information into small text plan tree templates (see Figure 6.8). The task agents integrate the trees from their children using coherence rules (see 5.5.2.1) for combining text plan trees. In doing so, the task agents first order the text plan trees

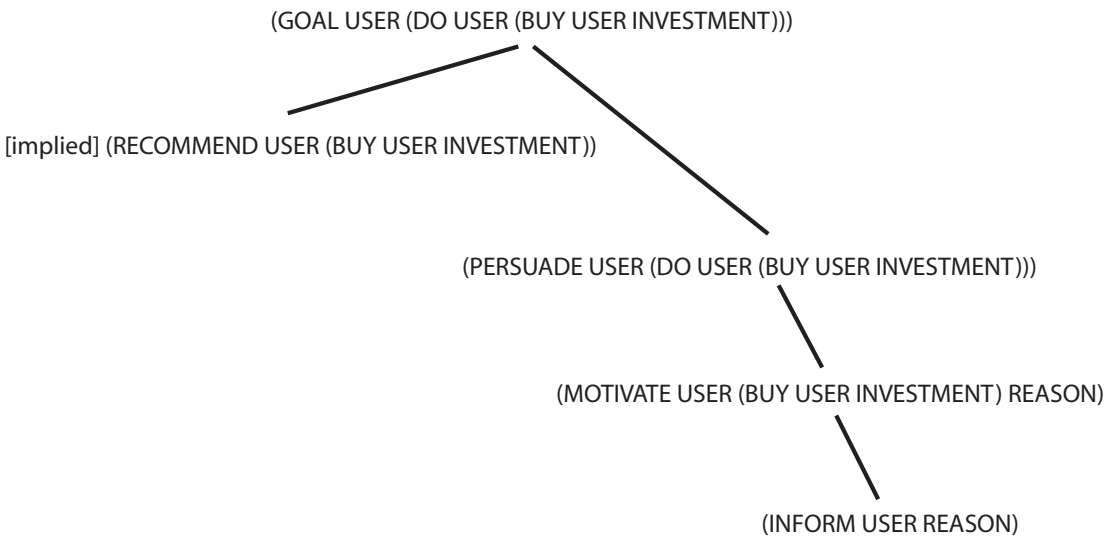


Figure 6.8: A simple text plan tree template.

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). The integrated text plan trees are then recursively propagated up the agent hierarchy. The final integration is performed by the Presentation Agent, resulting in a single tree. That tree is resolved to text via templates, and the text is presented to the user.

It is important to note that *MADSUM* does not seek to provide an "optimal" solution that could be determined by simultaneously considering all possible assemblies of all possible subtrees (see Section 5.5.1 for other approaches to optimality and alternatives). 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

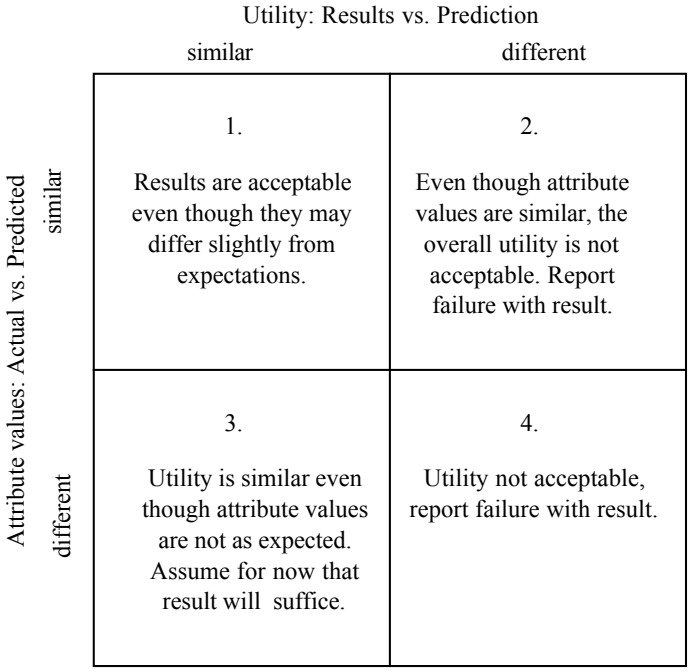


Figure 6.9: Categorizing results that vary from expectations.

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.

6.6 Managing Failure

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. For *MADSUM* a "failure" is 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; for example, in the financial decision support domain a message that does not meet expected utility because it exceeds a soft constraint on length may still be of more value to a user than no message at all. Also, a partial result that has below expected utility when considered by itself may be acceptable

in the context of other agent results with which it can be integrated..

While agents can predict the attributes of a result that they will provide, the responses of their sources change over time, and can change between the time attributes are predicted (during the negotiation process) and the time that results are produced and propagated up the hierarchy. Thus result attributes may in fact be very different from predictions, and as a result the utility may be different as well, potentially resulting in failure. The four possible combinations of possible utility and attribute[8] results are shown in Figure 6.9. The simple cases are those numbered 2 and 4, when utility is not similar to expectation. Both of these cases require a *MADSUM* agent to manage the results as a failure. However, case 3 is interesting because *MADSUM* focuses on expected utility, not expected attribute values. In this case *MADSUM* can avoid, for now, attempting to correct a result that has attributes that do not match the original bid. In circumstances where the differing attribute values cause difficulties later, the protocol below describes how such eventual failures are managed.

In addition to failures that can result from a dynamic environment, the resource allocation issue described in Section 6.4.5 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 the occurrence of failures 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. Thus failure is a part of the normal operation of *MADSUM*, and the architecture has been designed to minimize performance penalties incurred while handling failure.

MADSUM has four core strategies for minimizing failure penalties:

1. *Task agents handle failures that occur below them whenever possible.* This strategy

[8] Attributes are referred to as a class here, but Figure 6.9 could also be applied to any single attribute and its relation to utility.

can prevent low-level failures from propagating up the agent hierarchy and overwhelming the top level agent with failure-related computation.

2. *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 strategy allows an agent to ask a child to fulfill an earlier offer without resorting to a communication-intensive bidding process. Agents also store all results received from children, even if those results were part of a bid set that is no longer being considered. If the current bid set results in failure, results from previous bid sets may be part of other bid sets considered, thus reducing the marginal cost of those sets.

3. *Agents 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.

4. *Failures that indicate substantial problems with a sub-agent's performance, such as a time-out or exceeding hard constraints, mean that the sub-agent's other bids will not be considered for failure recovery purposes.* This strategy reflects the assumption that an agent will not produce un-usable results or fail to respond unless something substantial is wrong (such as a network failure). Thus once an agent A fails to communicate or produce useful results, that agent's parent should not plan to make use of A's capabilities (bids) for at least the duration of the current task.

These four strategies are implemented in the failure management protocol.

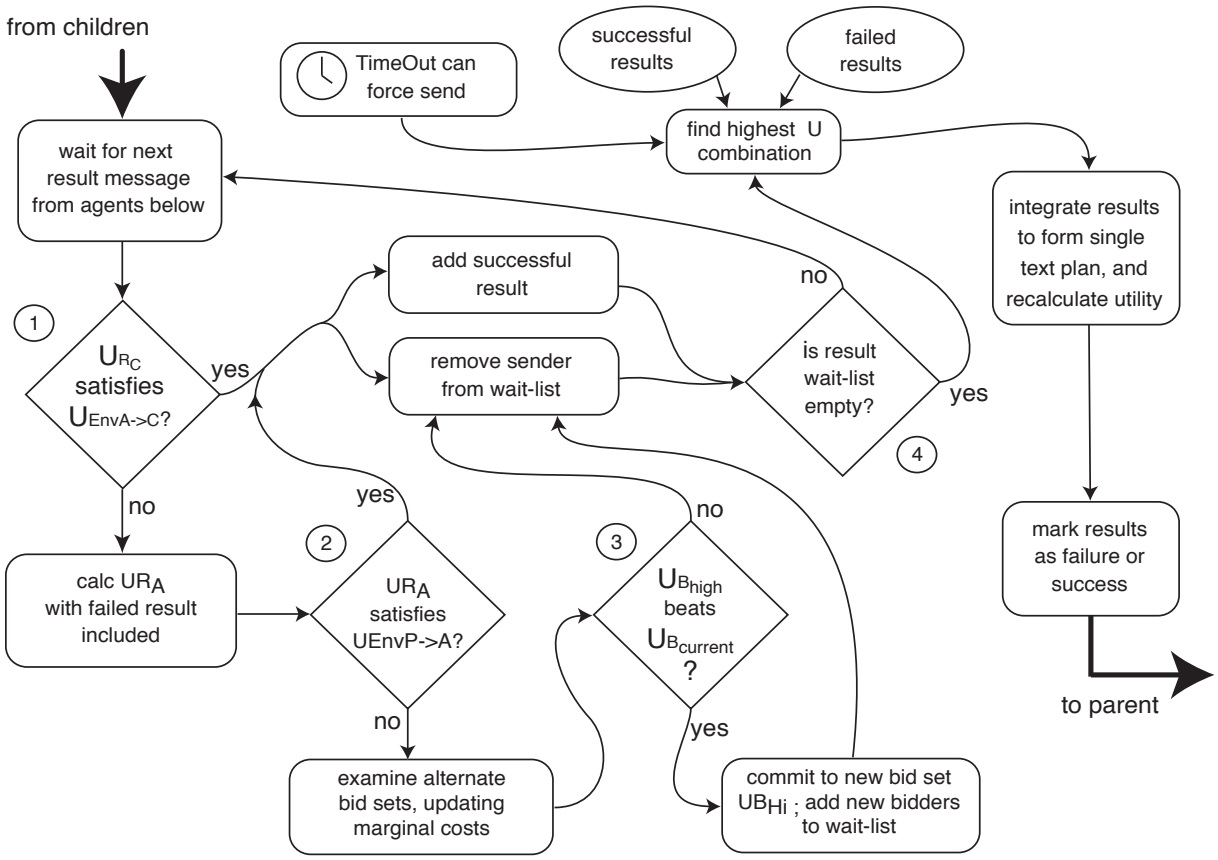


Figure 6.10: The *MADSUM* protocol for handling failed results.

6.6.1 The failure management protocol

Figure 6.10 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 examining a result's existing attribute values, and thus do not require extensive calculation or extensive domain knowledge (as described below).

Consider an agent A with parent P and child C. 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 $U_{EnvP \rightarrow A}$. The numbered items below correspond with the numbered diamond-shaped decision points in Figure 6.10.

① The result from the child C, R_C, has utility U_{R_C} and is compared to $U_{EnvA \rightarrow C}$ (i.e. the envelope that A had sent to C after the negotiation process) to see if it has failed. If U_{R_C} *does* meet the minimum utility described in $U_{EnvA \rightarrow C}$ then R_C is a successful result and the child sender C is removed from A's list of pending results ("yes" from decision point ① in Figure 6.10).

If U_{R_C} does *not* meet the minimum utility described in $U_{EnvA \rightarrow C}$ then R_C fails ("no" from decision point ①). However, it is possible that while this child's result does not meet the expectations of the envelope, the combination of all children's results[9] may still meet the overall utility required by P, i.e. $U_{EnvP \rightarrow A}$.

② The agent checks to see if U_{R_C}, though lower than expected, still allows all child results (either actually received or expected) to be combined such that U_{R_A} meets or exceeds $U_{EnvP \rightarrow A}$. If this is the case ("yes" from decision point ②), 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.

If the failed result from C prevents the expected U_{R_A} from meeting $U_{EnvP \rightarrow A}$ ("no" from decision point ②), then 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 some part of B_i, then the concept of marginal "cost" must

[9] This is the utility of the combined results, not the sum of the utilities of the results, since overall utility must be calculated on the separately combined attributes.

be considered with respect to each of the resource attributes since previous commitments to bids may reduce the amount of additional (marginal) resource outlay required for a given B_i.

The marginal expenditure of a resource necessary to commit to the bids in some new B_i depends on the resource in question. Consider dollar cost, one of the resource attributes that must be considered in the search for an alternative bid set. Suppose that A has committed to the bids in bid set B_{old}, of which b_k is one bid. Once the agent has committed to a bid b_k, the source agent is entitled to dollar compensation (having labored, or incurred expense), and so that money is already spent, and not using the result will not gain a refund. Now suppose that some other bid b_f in B_{old} produced a failed result, and that A must abandon B_{old} because there is no expectation that it will produce the desired utility. Three things are noteworthy at this point:

- A has already paid for the result from b_k (and the other bids in B_{old}), whether or not that result has been delivered yet;
- A's dollar resources for this task have been depleted by (at least) the cost of b_k, and so A has fewer resources now to consider alternative bid sets which might produce sufficient utility;
- If A has an alternative bid set B_{new} that includes b_k, then because A has already paid for b_k, the *additional* resource expenditure by A (i.e. the marginal dollar cost) to secure B_{new} is reduced by the dollar cost of b_k.

Consideration of marginal resource expenditure makes bid sets that use previously agreed upon bids more attractive. Note, however, that not all resources behave like dollar cost. In particular, there is no point in viewing the resource attribute "length" in this way, since simply not using a result avoids the length of that result. The resource attribute "time" behaves in yet a third way, since the marginal cost of a

previously chosen bid is the amount of time *remaining* to complete that bid, while a newly chosen bid will take its full expected time.

After adjusting the expected resource outlays for each B_i based on the expected *marginal* resource expenditures for each bid, A re-calculates utility for each B_i.

③ The agent examines the newly calculated utility of each bid set B_i and selects the bid set with the highest calculated utility, B_{high}. $U_{B_{high}}$ is compared to the (now lowered) expected utility of the bid set $B_{current}$ (which contains the bid for the failed R_C). If $U_{B_{high}}$ is *lower* than $U_{B_{current}}$, then the best strategy is to continue the present course ("no" from decision point ③). Otherwise, if $U_{B_{high}}$ is greater ("yes" from decision point ③), then A will commit to the parts of B_{high} to which it has not already done so.

④ The agent checks to see what results are still pending. The expectation of a result is added to the wait-list when the agent commits to each bid in a bid set, and the expectation stays on the list until the result is received, or a previously determined deadline is reached (a "timeout" limit).

Finally, when all expected results are processed (the wait-list is empty) or the time-out is reached, the highest utility combination of results is integrated to form a single text plan. The results are then marked as either meeting $U_{EnvP \rightarrow A}$ or failing. Results are sent to the parent P even if they fail, so that P can make use of the result if P finds that the result increases overall utility. The only circumstance in which no result would be propagated to P would be if all available results violated hard constraints (see Section 6.4.4).

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

If every agent fails, but submits failed results, the Presentation agent will present the highest utility message possible given the results available. Utility is a way for the *MADSUM* system to estimate the worth of a message to a user; it is not directly related to the user in any way. For example, the user does not specify some minimum utility for a response.

If no agent submits a result (for example, if all results violate a hard constraint, or there is a catastrophic network failure and all agents reach timeouts) then the Presentation Agent reports that it was unable to construct a message for the user's request.

6.6.2 Failure Protocol Complexity

Communication between agents is an expensive part of any multi-agent system. After committing to a particular set of bids of size n, an agent expects n replies with results. Next I examine how the number of agent responses can vary from n when a result fails to meet the expected utility.

For any task agent A making choices about failure recovery, the number of options that A has is finite under the failure protocol. Suppose A is currently considering a failed result from child C. The result R_C was to be part of a set of results delivered according to the set of child bids $BidSet_1$ of size n_1 to which A has committed. A has the following possible options (see Figure 6.10) in the failure protocol:

1. stay the current course and assume that the remaining utility derived from all results of $BidSet_1$ will be sufficient.

2. choose an alternate set of bids $BidSet_{alt}$ from children and communicate with children about it; or

3. decide that A has failed and report a failure to A's parent.

If the number of alternative sets of bids is s_{alts}, then option 2 can happen s_{alts} ways and thus the total number of options is $2 + s_{alts}$.

Now suppose a result from a child is unsatisfactory or missing, i.e. it fails, and agent A can respond with any of the three options. I will examine how many agent communications result from each option.

If the agent decides to stay the course (option 1), then it could also do so once for each other bid in the current $BidSet_1$, for a maximum of $|BidSet_1|$ or n_1 times, resulting in the same number of messages originally expected.

If the agent chooses option 2 and pursues some $BidSet_{alt}$ of size n_{alt} that would meet expected utility, then the handling of this particular failure of R_C is over, *and* the number of alternative bid sets s_{alts} is decreased by one. Thus the maximum number of times this strategy can be pursued is s_{alts}. In this case, the originally expected agent communications are still expected, and additionally n_{alt} commitments could be sent and n_{alt} results expected. However, the protocol favors choosing bid sets that overlap (due to lower marginal cost), and so it will not be n_{alt} communications going out and back, but rather a *marginal communication cost* representing only the bids in the new bid set that have *not* previously been attempted:

$$n_{marginal} = |BidSet_{alt}| - \left|BidSet_1 \bigcap BidSet_{alt}\right|$$

Finally, option 3, choosing to report failure to A's parent can only be done once and requires one message.

6.6.2.1 Worst case failure scenario

In the worst possible case, with all results failing and A not abandoning a bid set until every result expected from it has failed, A will alternate between options 1 and 2 as follows. First, A will choose to stay the course for every bid in the current bid set (option 1) until the last bid's result fails (incurring a time penalty for waiting for results, but no additional communication). Then after that bid set is exhausted, A will choose an alternate bid set (option 2, incurring a communication cost the size of each bid set, since in the worst case there will be no overlapping bids) and repeat. If this worst case were

to occur, then from A's perspective the protocol would have worst case communication complexity of

$$\sum_i |BidSet_i|$$

for all alternative bidsets.

However, this worst case is unlikely in practice, since *MADSUM* is designed to avoid it. First, setting an appropriate time limit will stop the system from exploring all possible options. If the time limit is not restrictive, then the system is free to pursue all options and may eventually explore a worst case scenario. Second, exploring options has resource costs other than time, and in *MADSUM* agents are also incurring dollar cost to explore options. Since the user can place hard constraints on dollar cost as well as time, this too can prevent worst case exploration. Ignoring the user's important ability to limit time and cost, I will consider the two possible cases, one where the user's utility function does not change, and one where it does, and show how the selection of a set of bid sets is designed to avoid the worst case.

The worst case is unlikely because if the utility function $U_{original}$ stays the same, then high utility sets are the most likely to provide utility within the envelope. Since high utility bid sets are likely to share lots of bids (see Section 6.4.3), cost will be far less than the sum of the size of the bidsets. Since $U_{original}$ stays the same, the alternative bid sets are likely to be unattractive and will not be explored since they probably don't provide utility within the envelope. If $U_{original}$ is replaced by some U_{alt}, then the high utility bidsets are either still the most attractive (in which case the previous reasoning applies) or the alternative bid sets are now more attractive. Because alternative bid sets are chosen to be not only distinct from high utility bid sets, but also from one another, it is unlikely that *many* alternatives will be attractive (i.e. will provide utility within the envelope). Thus even though alternative bid sets are more likely to have distinct bids in them, few bid sets are likely to be explored.

Thus the circumstances under which the worst case *would* occur would be when

- the time and cost limits are very high (indicating that the user doesn't mind spending large resources on exploring possible message space); *and*
- the utility of all the bid sets is approximately the same, so that any one of the high or alternate bid sets meets the envelope;
- none of the bid sets share bids, despite the fact that they all have similar utility; *and*
- no task agent fails until every one of its children has failed.

6.6.2.2 A likely failure scenario

A likely failure scenario, if large numbers of child result failures are occurring, is that A will run out of time and send a TIMEOUT message[10], giving A's parent the opportunity to try to replace A's result with results from other agents. Typically (see Section 6.7) as failures occur, and before time runs out, A will go through the available alternative bid sets, but at a marginal communication cost of only one message per bid set. This is because all the top utility bid sets tend to differ by only one component bid, i.e. if A chooses a bid set B_{high} containing $C_1 bid_a$, $C_2 bid_a$, and $C_3 bid_a$, then A's next highest utility bid set B_{alt}, might contain $C_1 bid_{\mathbf{b}}$, $C_2 bid_a$, and $C_3 bid_a$. Thus the marginal communication cost of changing to B_{alt} will only require one new message to C_1. (If the bid sets differ by more than one bid, they are likely to be so dissimilar that they would not all meet the required utility.)

Since the marginal communication cost of changing bid sets is likely one, the cost to A of all of A's children failing is likely linear in the number of bid sets A has to choose from. Again, this probably does not include bid sets generated for alternate utility circumstances, since they would not meet expected utility. But in any case, since

[10] All agents in the hierarchy calculate how much time they have to work on a task when they first receive a message allocating them resources for that task. If the time limit is reached before the agent completes the task, the agent sends a TIMEOUT message in lieu of results.

the number of bid sets is relatively small (the system default is three high utility bid sets and three alternate utility bid sets), a failure response that is linear for the number of bid sets is very fast.

Still, even this linear complexity assumes that all children are failing to meet expected utility. A more expected circumstance will be some agents each experiencing some children with some failed results. In the special case of an agent that has a child report a single failure (after which the child would succeed), the expected marginal communication would be one (since the follow-up communication from the parent would succeed). Therefore if multiple agents had children with single failures, the expected marginal communication cost would be linear in the number of failures. This is shown experimentally in Section 6.7, where the increase in total run time increases linearly whether the incremental failure occurs under an agent already experiencing a failed child result (which is expected to be linear as described above) or the incremental failure occurs under an agent whose other children are successful (which is expected to add one communication, or be linear across the system).

6.7 Evaluation

The system takes about 17 seconds for the two full rounds of communication and text tree processing if all thirteen agents are running on a single G4 laptop. Processing a complex problem with multiple information failures can take close to a minute. These figures can be reduced substantially when the parallel nature of the architecture is leveraged, and when full Java runtime optimization is turned on. Optimization has been turned off for MADSUM timing runs since the optimization effects are cumulative over a series of runs, rendering consecutive runs incomparable.

I evaluated the performance of the architecture in three directions. First, I hypothesized that while the execution time of the system was exponentially related to the degree of the agent hierarchy (i.e. the maximum number of children an agent can have), that within a given degree the addition and operation of new agents can be accomplished with

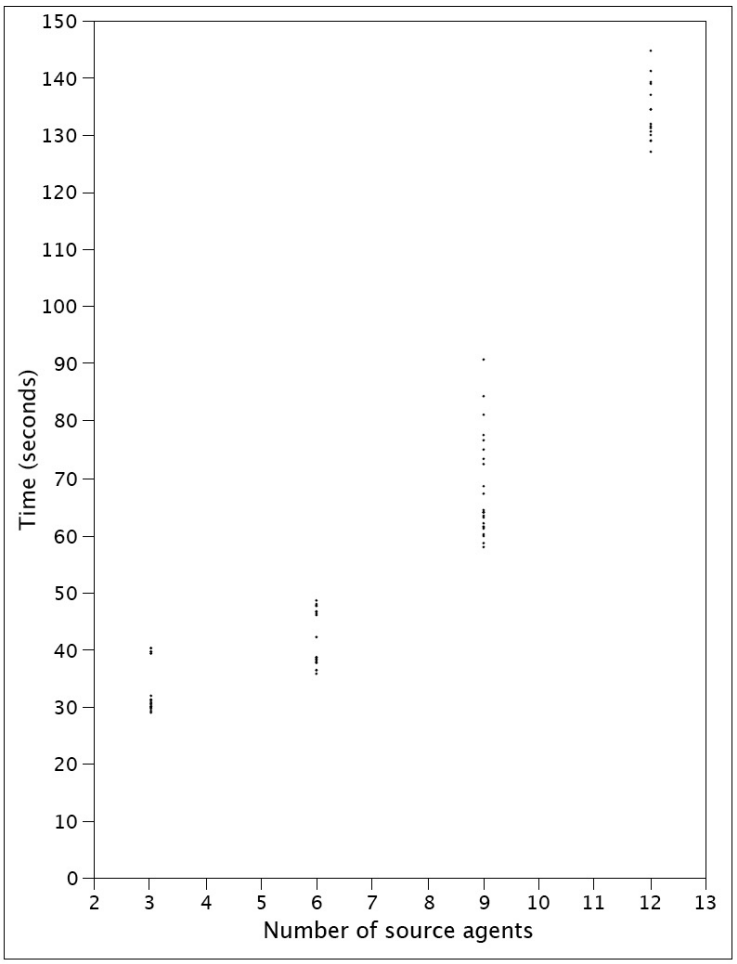


Figure 6.11: Execution time vs. number of source agents, on three machines

only a linear penalty. The experiments to test this hypothesis are shown in Section 6.7.1. My second hypothesis was that the system, when faced with failed results, would behave as predicted in Section 6.6.2. This required the performance of controlled tests of the failure protocol to observe its behavior (see Section 6.7.2). Third, I hypothesized that the system was capable of taking advantage of operating in parallel by distributing work across agents on different machines and so reducing overall execution time, and so I devised testing to evaluate behavior when the system was deployed across three separate machines. The results are described in Section 6.7.3.

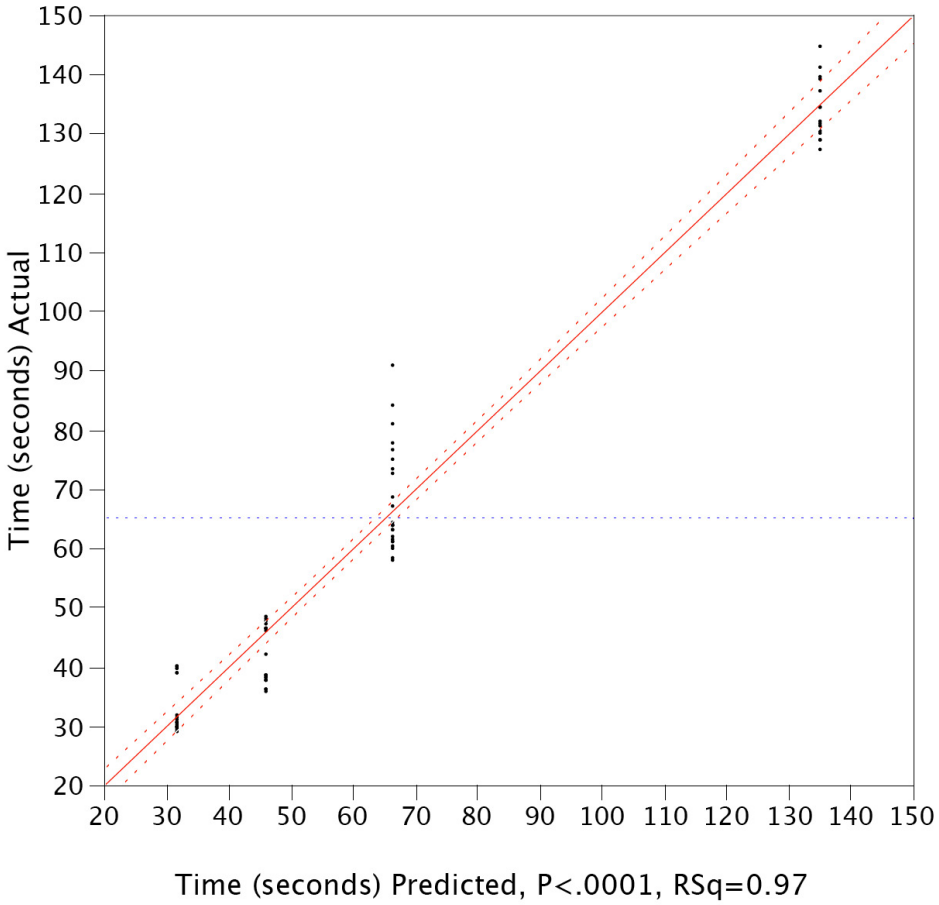


Figure 6.12: Execution time vs. Predicted Time, showing that increasing the number of children per task agent increases runtime according to Equation 6.3. The grouped data show, from left to right, one source child per task agent through four source children per task agent.

6.7.1 System performance as agents are added

To confirm my first evaluation hypothesis, I must show that the system's execution time is exponentially related to the degree of the agent hierarchy, and then must demonstrate a linear penalty for agents added within a given degree (where degree is the maximum number of children an agent can have).

$$(m+1)^n - 1 \tag{6.2}$$

Equation 6.2 (a repeat of Equation 6.1 from page 137 where m is the maximum number of bids submitted by a child agent, and n the number of children) notes the relation between the number of children of an agent and the complexity of the bid analysis. The graph in Figure 6.11 shows the raw data generated by running between three and twelve source agents under a constant three task agents and one Presentation Agent. First three sources were run, one as a child for each of the three task agents, then two children per task agent, etc., up to four children per task agent. Each of the three task agents was operating on a different machine, i.e. in parallel, so that the execution time curve shown is not representative of the work being done by all three task agents, but only of the work done by (approximately) a single task agent. For this experiment between seven and sixteen agents were evenly distributed across three Apple computers: an 867 MHz laptop with 512 Mb RAM, a 1.3 GHz laptop with 1.25 Gb RAM, and a 800 MHz desktop with 384 Mb RAM. The Presentation Agent was run on the 867 MHz laptop in addition to the task agent being run there.

This data should fit Equation 6.2 on page 166. More specifically, in this case where each task agent had n children and a total of eight bids (five high utility and three alternative utility bids) per child, the number of combinations considered by a task agent is

$$(8+1)^n - 1 \tag{6.3}$$

I then assumed that the number of combinations created and examined was closely related to execution time, which will be true for large numbers of combinations. Figure 6.12 shows the actual time data correlated with the predicted time result from Equation 6.3. The horizonal line is the mean of the actual data projected across the graph for reference. The diagonal line is where all data points would be if they perfectly corresponded with the predicted value. The $f = 0.95$ confidence curves on either side of the diagonal line are there to show how well the actual data fit the *mean*, i.e. if the data were mostly random and could be predicted by averaging, then the mean line should fall within the

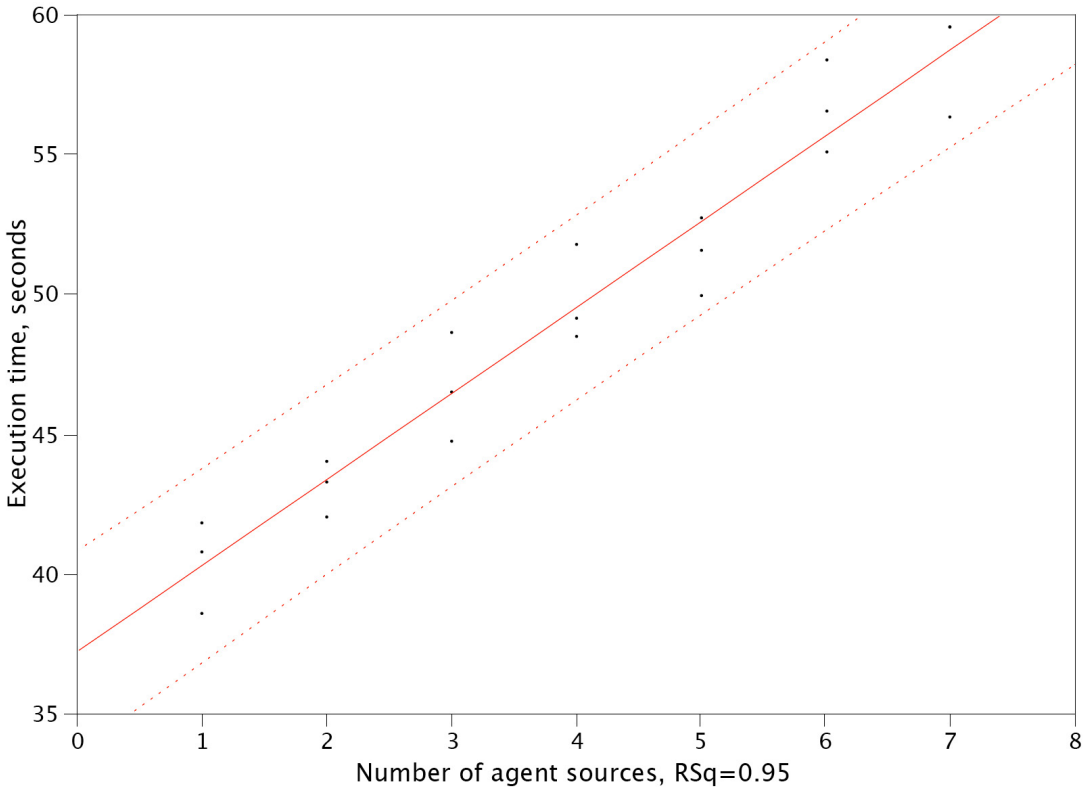


Figure 6.13: Runtime vs. sources count, 42 trials

confidence curves. Clearly the data here cannot be predicted by a simple mean, since the mean line does not come close to fitting within the confidence curves, as numerically shown by the value $P < .0001$. In fact, the data is not only not random, but highly correlated to Equation 6.3. The R-squared for the correlation is 0.97, indicating that 97 percent of the observed variance from the overall data mean can be explained by the predicting formula.

Since multiple *MADSUM* agents are required to perform a decision support task, it was my intent that the design should scale well. This would seem to conflict with the data shown in Figure 6.11; however, *MADSUM* scales well as long as the degree of the hierarchy is preserved. My hypothesis was that once the maximum number of children an agent may have is fixed, adding agents to the system within that number results in only

a linear performance penalty. My hypothesis is confirmed by the results shown in Figure 6.13, which shows that incremental additions of source agents to the system result in a linear increase in time, provided the degree of the tree is not increased by this addition. The graphed diagonal line is the best linear fit, and the dotted lines are drawn at $\alpha = .05$ for the data points. The R-squared value is 0.95, indicating that 95 percent of the observed variance from the data mean can be explained by the linear prediction shown. For this experiment, between seven and thirteen agents ran on a single Apple 867MHz laptop with 512 Mb RAM. Data are shown for an agent hierarchy of degree three (since the PA has three children). All six task agents are active throughout the test (the Presentation Agent, RISK, VALUE, GOAL, DEBT RISK, and INDUSTRY RISK), but the number of source agents (the seven information agents at the leaves of the hierarchy, children of task agents) varies from zero to seven.

6.7.2 Evaluating performance with failures

My second hypothesis was that the system would behave as predicted in Section 6.6.2 if failing results were introduced. I performed controlled tests of the failure protocol, varying the number of failures, to observe *MADSUM*'s behavior. As explained in Section 6.6, it was my intent that the failure recovery mechanism be used by the system to ameliorate poor choices made with good intent, but which then proved suboptimal under dynamic conditions. This requires that the mechanism not be overly burdensome.

I chose to test the failure mechanism in two ways. First, I wanted to demonstrate that within certain parameters described in Section 6.6.2, i.e. within *MADSUM*'s bid protocol, that typical failures exact a time penalty that is essentially linear. Second, I wished to demonstrate that the mechanism is indeed distributed and able to benefit from parallelization, and that failures can be handled in parallel when task agents are operating on more than one system.

Figure 6.14 shows the relationship between the number of agents returning a failed result and execution time when running on a single machine. For this experiment fourteen

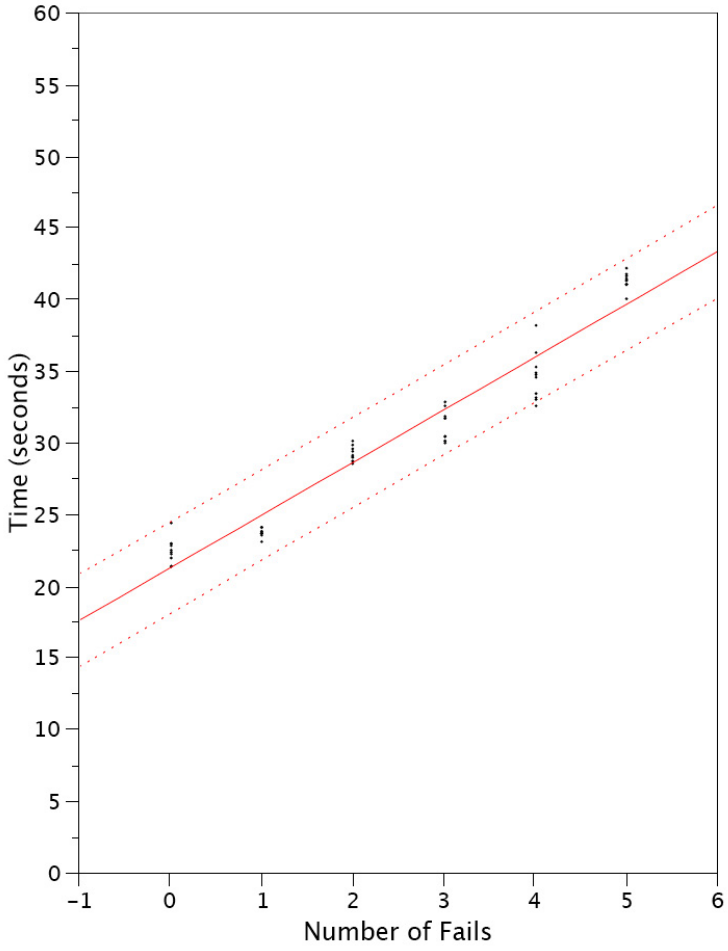


Figure 6.14: Number of agents sending one failed result each vs. Execution Time. Linear fit shown with R-square of 0.94. Dotted lines are .95 confidence boundaries.

agents ran on a single 867MHz laptop with 512 Mb RAM. The experiment consisted of 60 trials with the thirteen agents previously described (six task agents and seven source agents, see Figure 6.2 on page 129), plus one reporting agent. Of the seven source agents, five[11] were given data that would result in the source returning a failed result. Of the five,

[11] The two source agents under GOAL were not selected to report failures since they calculate initial bids dynamically from portfolio data, i.e. their bids are based on their portfolio data, so to create a failure would have involved modifying a file of portfolio data during execution so that the portfolio changed between bidding and

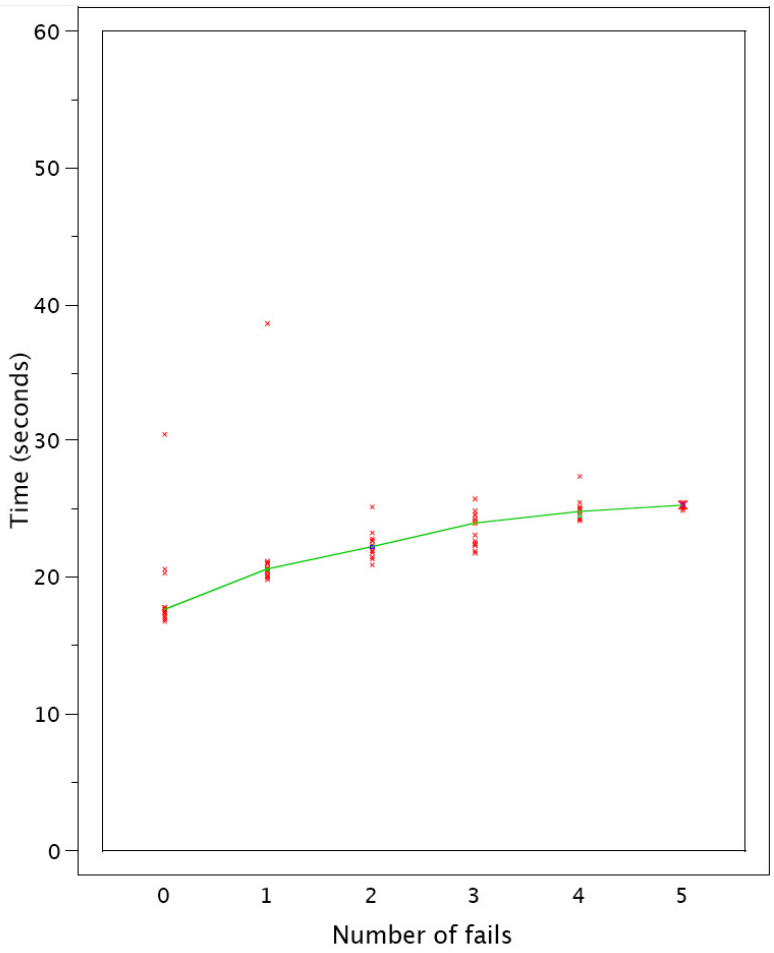


Figure 6.15: Number of agents sending one failed result each vs. execution time, with agents distributed across three machines. Line connects median values for each data group. Two outliers were due to observed communication delays.

three were under RISK (CR, DE, IR) and two were under VALUE (PE and RE). These five sources also had alternative data that would *not* result in a failure, so that when they were contacted a second time during the failure protocol they would return a successful result with utility close to that in the original bid. The experiment started with no source agent using the "failure" data, then one source, then two, etc., until five sources were all returning failed data (the horizontal axis of the graph). The vertical axis represents

---

the return of results.

execution time from when the PA receives the first message to the final output of text.

The relationship between the number of agent failures and execution time is shown to be linear with an R-square of 0.94, which confirms my second hypothesis. Note that this is not the worst-case failure scenario described in Section 6.6, but instead just a bad case scenario (since up to five of seven source agents are failing).

6.7.3 Evaluating parallel speed-up

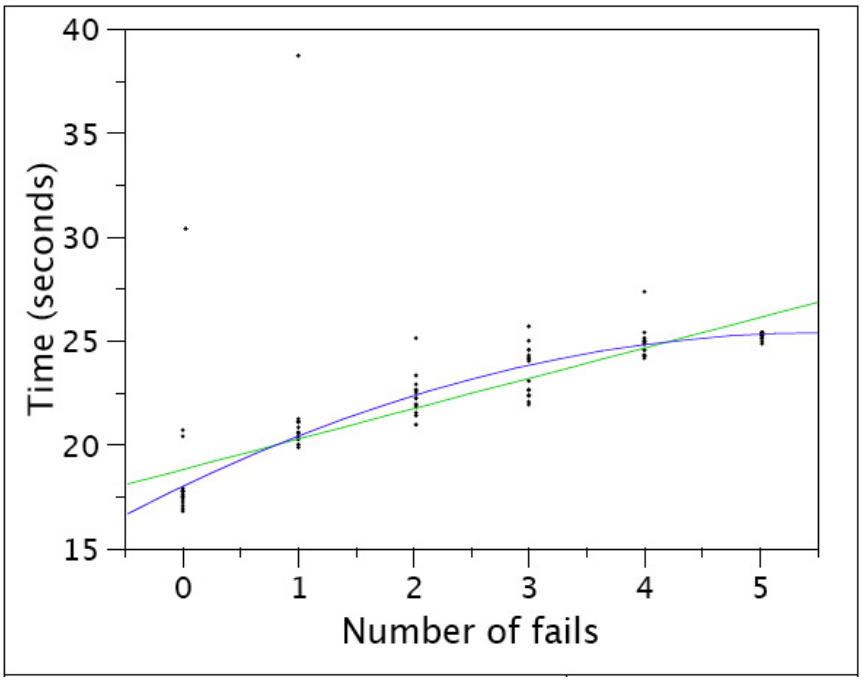


Figure 6.16: Number of agents sending one failed result each vs. Execution Time. Linear fit shown with R-Square of 0.85; 2nd degree polynomial fit shown with R-square 0.90. Two outliers were due to observed communication delays.

My third hypothesis was that the system was capable of taking advantage of operating in parallel by distributing work across agents on different machines, resulting in reduced execution time. To test this I performed the failure evaluation from Section 6.7.2 again, only with agents operating in parallel on multiple machines.

Figure 6.15 shows results from an experiment similar to Figure 6.14, but this time the agents are distributed across three machines (the same machines described in Section 6.7.1). As one might expect, the linear increase in time is gone, since the time penalty associated with handling an increasing number of failures can be hidden by the process occurring in parallel (The three agents RISK, VALUE, and GOAL are operating on different machines). Here a line connects the median points of each data group, showing the decreasing penalty on overall time as the maximum number of failures on a single machine, three, is reached at four fails, when CR, IR, and DE have all failed under RISK. The two data outliers are real network communication errors, where a message sent by one agent was not received by another, thus invoking a timeout penalty. Figure 6.16 shows that a second degree polynomial (R-square 0.90) is in fact a better fit for the data than a line (R-square 0.85), confirming that when the failure protocol is operating in parallel the penalty is bounded by the agent with the most children.

6.8 Summary

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 of bids from each agent are presented to a parent agent, who selects multiple baskets of bids from the child agents. Baskets represent different combinations of bids to provide different utility to the user for varying resource commitments. If the user's utility function changes, or certain results are not as predicted, the variety of the baskets provides alternatives without having to renegotiate. The bid selection process moves recursively up to the Presentation Agent, where the highest utility basket (under the current utility function) is chosen. Starting with the Presentation Agent, *MADSUM* agents recursively allocate expected utility to sub-agents, allowing the sub-agents to determine precisely how to derive that utility themselves (within optional constraints). This mechanism provides agents with flexibility to operate in a dynamic environment. Changes in the environment

or in the user's utility function cause information results to fail to meet expected utility, triggering a fast failure management protocol that utilizes stored information about previous bids and pre-calculated alternatives.

All timing experiments support my hypotheses about *MADSUM*'s performance and confirm the complexity analysis. In particular, while increasing the degree of the agent hierarchy tree is exponentially expensive, testing shows that

- Adding new source agents only increases execution time linearly on a single processor when the degree of the hierarchy tree does not change;
- The system benefits from parallelization, with the execution time penalty associated with adding agents being sub-linear when the degree of the hierarchy tree does not change;
- *MADSUM*'s efficient failure management protocol can handle certain failures in linear time – in particular, those failures that are expected to arise as a result of a dynamic information environment.

Chapter 7

CONCLUSIONS AND FUTURE WORK

This thesis is the result of two related investigations into areas of communication in decision support. Both explored the communication planning problems found when decision support information provided to a system comes from multiple, independent sources. The first investigation resulted from a need to improve the quality of messages designed for use by a physician in a trauma setting, and led to development of the implemented system *RTPI* (Rule-based Text Plan Integrator). The second investigation was the result of the application of some principles drawn from *RTPI* in a fully distributed, implemented decision support system, *MADSUM* (Multi-Agent Decision Support via User Modeling) designed to function in a dynamic information environment. This chapter summarizes the research and suggests areas for future work.

7.1 RTPI

RTPI was developed in response to a need for improved output from the trauma critiquing system, TraumaTIQ [WCC+98, GW96]. TraumaTIQ generates messages to aid a trauma physician in making real-time decisions about a trauma case in an emergency room setting. These messages were each coherent and concise when considered individually, but usually appeared in sets. When the message sets were viewed as a whole, they appeared to be redundant, and occasionally incoherent. These characteristics are not desirable in a setting where rapid assimilation by the physician is critical to success. *RTPI* was designed to examine a *set* of message text plans and remove any inconsistencies

that arose from the manner of presentation, and to increase set conciseness by removing redundancies where possible.

RTPI draws on rhetorical structure and discourse theory to incrementally improve a message set by successively integrating individual text plans when the plans share subject matter. *RTPI* uses transformational rules to aggregate and manipulate text plan trees according to the communicative goals they achieve. The integrated messages that result from application of the rules must retain *all* of the communicative goals of the original messages, and arrange for the expression of these goals. In other words, *RTPI* does not have the freedom to abandon a communicative goal to improve conciseness or coherence, since every goal has been determined by TraumaTIQ to be important to the case at hand.

RTPI's improved message sets were never tested in the environment for which they were intended, due mostly to issues related to deploying TraumAID and TraumaTIQ. (They both required rapid, real-time input of trauma case information, which proved elusive [WCC+98].) However, it did prove possible to test aspects of the messages without recreating the trauma environment (see Section 3.4). Comparison of the message sets generated by *RTPI* and those generated by TraumaTIQ showed significant reduction in the total number of messages in the sets, as well as reduction in the number of times that the system had to mention various medical actions and goals. Comparing coherence is more subjective; three human subjects making pairwise comparisons all preferred *RTPI*'s messages over 90 percent of the time, and strongly preferred them 69 percent of the time.

The design and implementation of *RTPI* generated several significant research results in the area of planning communication for decision support. In particular this work has shown that:

- analyzing classes of messages in terms of their rhetorical structure allows the development of integration rules that work on such structures;

- these integration rules can be used to reorganize sets of messages to reduce repetition and improve readability, while preserving all of the original messages' communicative intent;

- when texts from independent sources are combined, the appearance of conflict can be created by the structure of the messages, even when there is no real semantic conflict in the underlying information, and integration rules can be designed to exploit the relationships between messages and prevent the appearance of conflict.

7.2 *MADSUM*

Continuing my investigations into systems that integrate input from multiple, independent sources, *MADSUM* is a fully distributed system of software agents where each source is a separate agent, potentially running on a separate computer system. *MADSUM* is designed to provide decision support in a dynamic information environment, while taking into account user preferences regarding resource use and message characteristics.

Decision-support is subject to many kinds of resource restrictions (e.g. dollar cost, text length, execution 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* assumes poor predictive models of ultimate information utility and thus requires dynamic organizational management in response to run-time information failures.

To address these circumstances and requirements, *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. A user model, including content preferences, deadlines, and cost and length constraints, informs both processes. *MADSUM* must weigh the benefit of different choices about resource usage and information selection. A multi-attribute utility function provides a decision-theoretic 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, *MADSUM* considers the priorities of the user with regards to the utility of the message, allowing *MADSUM* to present high-utility messages about poor investment choices which have low utility to the user.

The major research contribution of *MADSUM* to decision support communication is that using the utility of a communication (instead of the utility of the contents) can create messages with content and ordering that are preferred by human readers when a hypothetical user has preferences about message presentation. *MADSUM* is the first decision support system to focus decision theory on the result of message generation about a choice, instead of on the choice itself. *MADSUM* also introduced new approaches to certain communicative planning problems. In particular

- *MADSUM* allows agents to commit to providing a measure of utility, instead of a list of specifications, thus allowing requesting agents to avoid costly resource distribution algorithms and providing agents with flexibility in how they achieve that utility.

- The development of an efficient failure protocol allows *MADSUM* to use result failure as a mechanism for handling problems that arise from allocating resources according to utility commitments.

7.3 Future work

RTPI was developed in response to a need in the trauma domain, but was designed to be domain-independent. Future work includes finding other domains like trauma decision support that involve an expert operating in real time and ordered actions to achieve multiple goals (machinists? chefs?), and then developing critiquing systems for these domains and investigating *RTPI*'s ability to function across domains.

While *MADSUM*'s content selection and ordering have been validated, the template realization is not as fluid, nor as flexible, as current realization techniques allow. Thus future work includes incorporating a full text/sentence planner into *MADSUM* to improve the range and variety of texts that can be generated.

MADSUM's responses have been evaluated in terms of ordering and selection. The next step will be to evaluate them in terms of effectiveness and usefulness. This requires that *MADSUM* be expanded into a fully operational web-based decision support system, with source agents that mine raw information from existing web sites. Then *MADSUM* could be incorporated into a stock game environment where players can use *MADSUM* to help with decisions, and evaluations could be both subjective, from the perspective of the user, and objective in terms of evaluated user portfolio performance.

A further goal is testing *MADSUM*'s reactions to dynamic environments by having humans rank *MADSUM*'s responses to a series of questions in changed environments. My hypothesis is that under limited time, removing alternative bid sets from *MADSUM* will reduce the quality of responses by crippling the failure recovery mechanism. The proposed web implementation could be used to document real cases of environment change and calculate computational savings due to the presence of the failure mechanism and alternative bid sets.

7.4 Summary

The two systems described in this dissertation are the result of investigations into the development of decision support system communications. *RTPI* explores the issues

surrounding integrating text from multiple independent sources in a real-time setting. In particular, *RTPI*'s rules show that when communicative plans from multiple sources are combined, careful integration strategies can reduce redundancy and improve conciseness, while preserving all of the original messages' communicative intent. Moreover, these rules demonstrate that while presenting texts from different sources together can create an appearance of conflict, application of well-designed integration rules can resolve such conflict.

MADSUM also explores text integration issues when multiple sources are involved, but focuses more on content selection and ordering when integrating (issues that did not occur to a significant degree in RTPI). Both of these tasks led to the development of a decision-theoretic user model that allows *MADSUM* to select and order content based on the utility of the resulting message to the user. The need to operate in a dynamic information environment also influenced the novel design choice to have agents commit to utility, rather than specific resource allocations, and the design of an efficient failure protocol to handle the information failures produced by this design decision. *MADSUM* is unique in its measurement of the utility of the *message* presented to the user, rather than simply measuring the utility of the recommendations contained in the message.

Appendix A

SCHEDULING INTERSECTIONS

TraumaTiq generates six kinds of scheduling messages. Their names and forms are as follows, where each ~a represents text to be inserted, such as the name of an action:

scheduling-urgent: "Caution: ~a before ~a because it is very urgent."

scheduling-priority: "Caution: ~a before ~a because it has a very high priority."

scheduling-site: "Caution: ~a before going to ~a to ~a."

scheduling-precede: "Caution: ~a before ~a because the latter may affect the results of the former."

scheduling-dependency: "Caution: do not ~a until after ~a. The outcome of the latter may affect the need for the former."

scheduling-precondition: "Caution: Remember to ~a before ~a."

The following scheduling messages all form intersections with omission messages and/or omission intersections in the 96 test cases:

- omission/scheduling-precondition
- omission/scheduling-urgent
- omission/scheduling-site

- omission/scheduling-precede
- omission/scheduling-dependency

Each of these types is handled by *RTPI*. Only scheduling-priority intersections with omissions are not represented in the 96 cases. However, the system does handle them should they occur.

Scheduling messages in TraumaTiq only warn of actions that should be performed *before* other actions that are already in the physician's plan. Thus intersections between omission and scheduling messages will always share the first action of the scheduling critique, since the second action of a scheduling critique must already be in the plan and cannot be the subject of an omission message.

Scheduling critique intersections of different types can also form intersections. Those (besides intersections of the same type of message) which *RTPI* handles in the actual trauma test cases are:

- scheduling-dependency/scheduling-site
- scheduling-priority/scheduling-precondition
- scheduling-precede/scheduling-precondition

Scheduling-priority/scheduling-site intersections are handled but do not occur in the data.

A.1 Scheduling messages and conflict

This section elaborates on Section 2.6.4. There are certain combinations involving scheduling messages which appear to have conflicting information. Analysis of the state of the planner shows that the information is consistent, but needs to be presented differently to avoid the appearance of conflict.

Only the combinations of scheduling-preconditions with errors of commission, or scheduling-preconditions with procedure-choice errors create this apparent conflict (a

Original TraumaTiq messages:
Giving analgesics as appropriate seems unmotivated because treating a compound fracture of the sternum has been proven to be unnecessary.

Please remember to check for medication allergies before you give analgesics as appropriate.

Integrated *RTPI* message:
Caution: giving analgesics as appropriate seems unmotivated because treating a compound fracture of the sternum has been proven to be unnecessary. However, if you still choose to give analgesics as appropriate, then remember to first check for medication allergies.

Figure A.1: Apparent conflict involving an error of commission with a scheduling error.

procedure choice error is generated when the system prefers an action over the one in the physician's plan). This is because, of the six scheduling error types, only scheduling-preconditions are generated from the physician's planned actions (which may be erroneous, in the system's view), while other scheduling errors are generated from examination of the system plan. Since commission and procedure choice errors are about actions the system won't plan for, only scheduling-preconditions ever share actions with commission and procedure choice errors.

In Figure A.1, a commit error tells the physician that "giving analgesics" is unmotivated, while a scheduling-precondition says "before you give analgesics as appropriate", thus implying that they will be given.

Procedure-choice errors can create a similar problem, as shown in Figure A.2. In these cases the list of actions must be the same, i.e. the system does not integrate these combinations for partially shared lists of actions. There were no partially shared lists of actions in the test cases.

Original TraumaTiq messages:
Performing local visual exploration of all abdominal wounds is preferred over doing a peritoneal lavage for ruling out a suspicious abdominal wall injury.

Please remember to check for laparotomy scars before you do a peritoneal lavage.

Integrated *RTPI* message:
Caution: performing local visual exploration of all abdominal wounds is preferred over doing a peritoneal lavage for ruling out a suspicious abdominal wall injury. However, if you do a peritoneal lavage, then remember to first check for laparotomy scars.

Figure A.2: Apparent conflict involving a procedure choice error and a scheduling error.

Appendix B

SUBJECT INFORMATION SHEET

The information sheet on the following page was given to subjects before they were given the selection and ordering questionnaires.

Information Sheet

You may refer to this sheet during the tasks. There are three tasks.

Each task involves comparing two sets of information. The sets are about an investment that is being considered by an individual.

The individual wants the information to be helpful in evaluating the investment. However, your job in the task is not to evaluate the investment, but to consider how well the information might help the potential investor (not you!).

Each pair contains information from three broad categories: an assessment of the investment's risk; an assessment of the investment's potential for growth; and how the investment relates to the investor's personal portfolio allocation goals (i.e. how investments are divided between different kinds of invesments, such as stocks (equities) vs. bonds).

Ratios: Financial ratios compare two or more numbers from a company's financial report. Investors use the ratios to evaluate an investment. For this task, it is only important to know whether investors generally prefer a ratio to be high or low when buying a stock.

Risk Ratios: estimate a company's ability to endure hard times. In investor parlance, an "aggressive" company behaves in a riskier way than a "moderate" or "conservative" company. Risk ratios often include some measure of a company's debt.

- Current Ratio: higher is better (i.e. a high current ratio implies less risk).
- Debt Equity ratio: lower is better.

Value Ratios: estimate whether a company's stock is a good value at the current price.

- Price Earnings ratio: lower is better.
- Return on Equity: higher is better.

Appendix C

MADSUM CONTENT SELECTION QUESTIONNAIRE

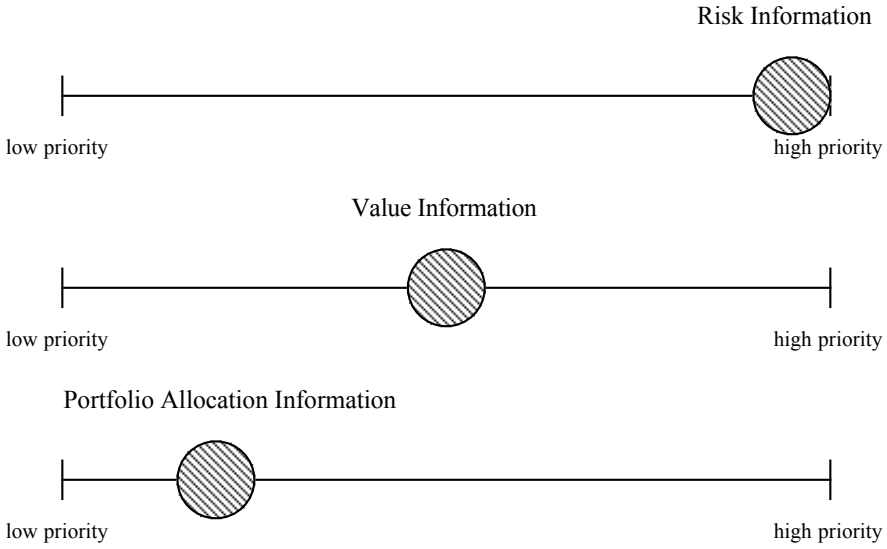
Content Task

This task presents several sliders on a page. The slider settings represent the priorities of an investor who is trying to make a decision about an investment. Under the sliders are two sets of information.

The order of the information is **not** important for this task. You are considering the content of each set, not the presentation.

Taking into account **both** the priorities of the investor and the importance of the information, decide which set's contents will be most helpful to the investor making a decision.

1. Taking into account **both** the priorities of the investor and the importance of the information, which set's contents will be most helpful to the investor making a decision?



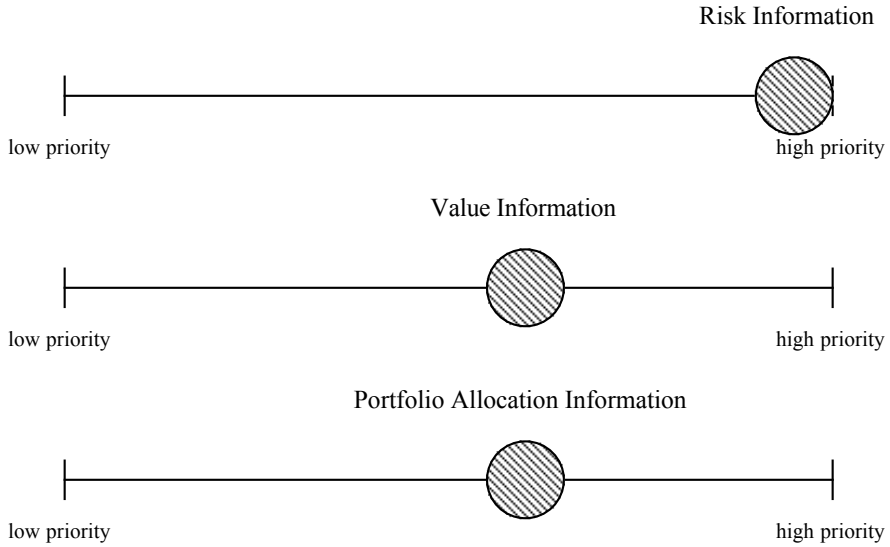
(a) company has a poor current ratio relative to the industry average
company's debt equity ratio is not good relative to the industry average

(b) company has a return on equity slightly better than the tech industry average
company has a history of moderate value metrics

Which set did you choose?

Briefly, why?

2. Taking into account **both** the priorities of the investor and the importance of the information, which set's contents will be most helpful to the investor making a decision?



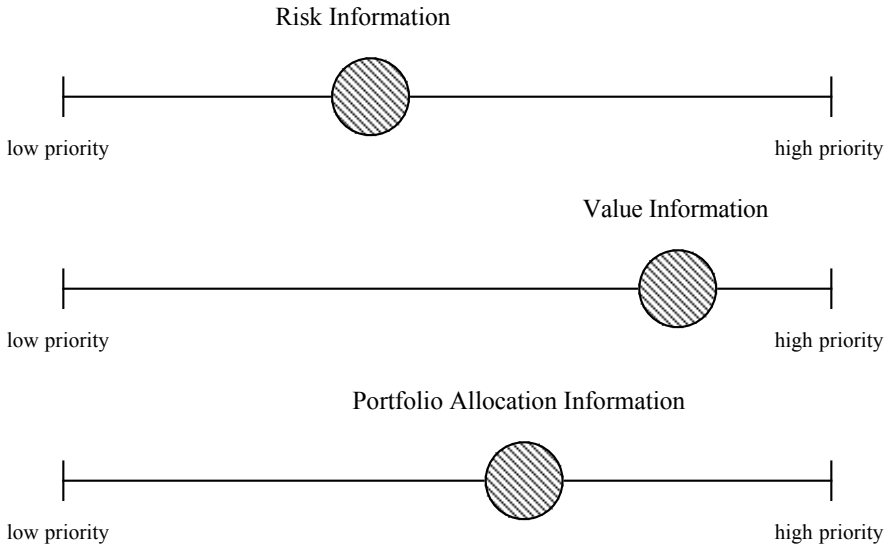
(a) company has historically maintained a moderate debt policy

(b) company has a very poor price earnings ratio, suggesting it may be over-valued

Which set did you choose?

Briefly, why?

3. Taking into account **both** the priorities of the investor and the importance of the information, which set's contents will be most helpful to the investor making a decision?



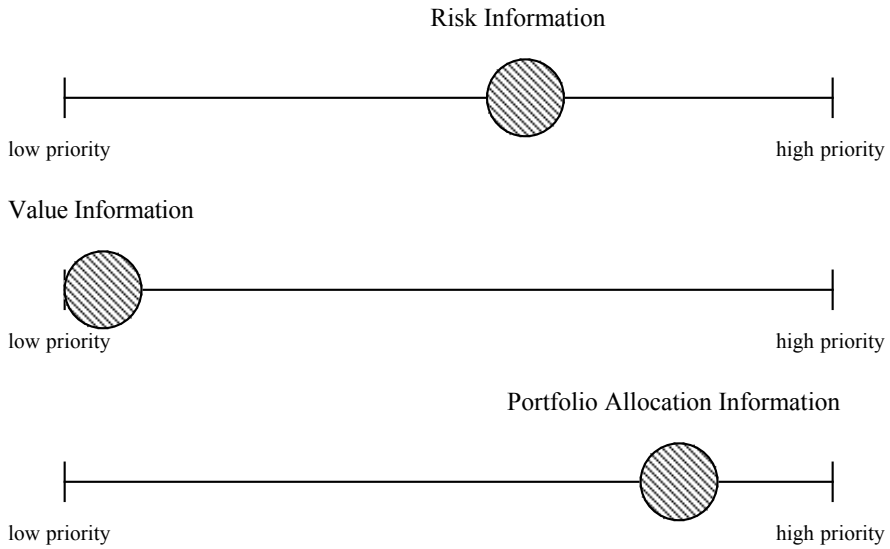
(a) company has a price earnings ratio similar to the tech industry average
company has a history of moderate value metrics
this investment is consistent with your stated allocation goal for equities

(b) company has a very poor current ratio relative to the industry average
company's debt equity ratio is not good relative to the industry average
company has a price earnings ratio similar to the tech industry average

Which set did you choose?

Briefly, why?

4. Taking into account **both** the priorities of the investor and the importance of the information, which set's contents will be most helpful to the investor making a decision?



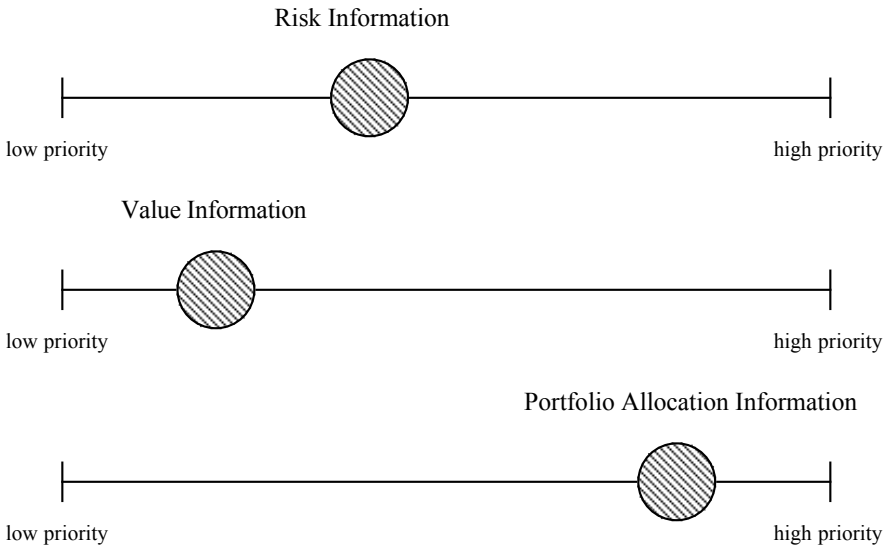
(a) company has a poor current ratio relative to the industry average
company's debt equity ratio is not good relative to the industry average
this investment is consistent with your stated allocation goal for equities

(b) company has a poor current ratio relative to the industry average
company's debt equity ratio is not good relative to the industry average

Which set did you choose?

Briefly, why?

5. Taking into account **both** the priorities of the investor and the importance of the information, which set's contents will be most helpful to the investor making a decision?



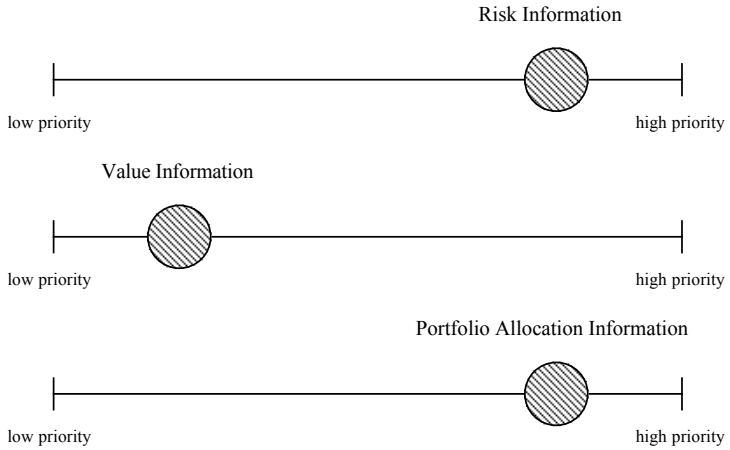
(a) this investment is consistent with your stated allocation goal for equities

(b) company has a very poor current ratio, indicating significant potential problems with near-term debt

Which set did you choose?

Briefly, why?

6. Taking into account **both** the priorities of the investor and the importance of the information, which set's contents will be most helpful to the investor making a decision?



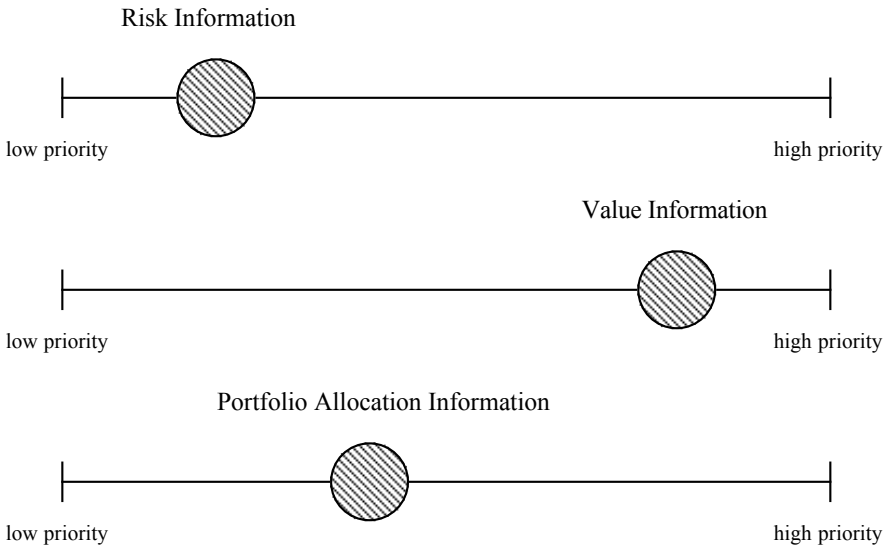
(a) company has a poor current ratio relative to the industry average
company's debt equity ratio is not good relative to the industry average
company has historically maintained a moderate debt policy
company has historically maintained a moderate risk profile
this investment substantially exceeds your stated allocation goal for equities

(b) company has a poor current ratio relative to the industry average
company's debt equity ratio is not good relative to the industry average
company has a return on equity slightly better than the tech industry average
company has a history of moderate value metrics

Which set did you choose?

Briefly, why?

7. Taking into account **both** the priorities of the investor and the importance of the information, which set's contents will be most helpful to the investor making a decision?



(a) company has a poor return on equity relative to the construction industry
company has a history of attractive value metrics
this investment is consistent with your stated allocation goal for equities

(b) company has a poor return on equity relative to the construction industry
company has a history of attractive value metrics

Which set did you choose?

Briefly, why?

Appendix D

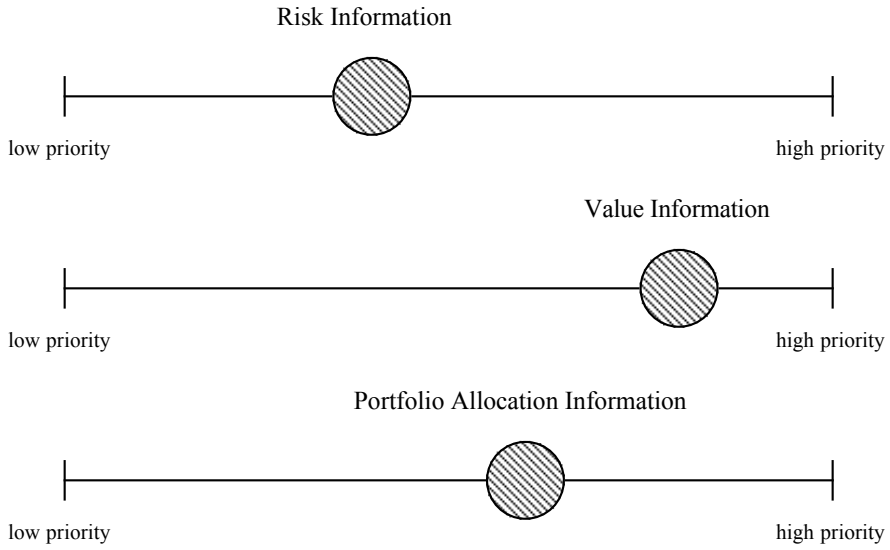
MADSUM CONTENT ORDERING QUESTIONNAIRE

Order Task

Each page has two paragraphs and a set of sliders. The slider settings represent the priorities of an investor who is trying to make a decision about an investment. Both paragraphs contain the same sentences, but the order is different.

Taking into account **both** the priorities of the investor and the importance of the information, decide which presentation order will be most effective at helping the investor make a decision.

1. Both paragraphs contain the *same* sentences. Taking into account **both** the priorities of the investor and the importance of the information, which presentation order will be most effective at helping the investor make a decision?



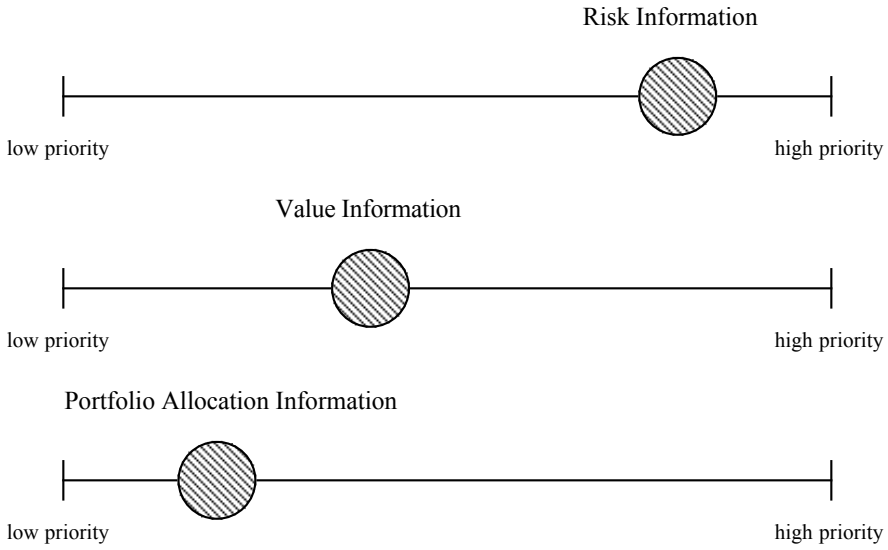
   (a) Portfolio metrics indicate this investment substantially exceeds your stated allocation goal for equities. From a value perspective the company has an acceptable price earnings ratio.

   (b) From a value perspective the company has an acceptable price earnings ratio. Portfolio metrics indicate this investment substantially exceeds your stated allocation goal for equities.

   Which presentation did you choose?

   Briefly, why?

2. Both paragraphs contain the *same* sentences. Taking into account **both** the priorities of the investor and the importance of the information, which presentation order will be most effective at helping the investor make a decision?



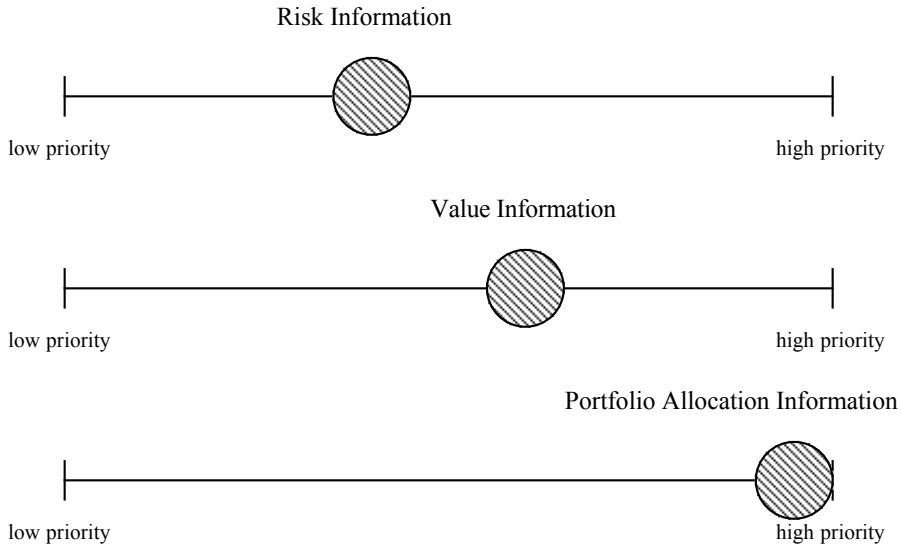
(a) Value metrics indicate the company has an excellent price earnings ratio, suggesting it may be under-valued. From a risk perspective the stock has a very strong current ratio, indicating good short-term liquidity.

(b) From a risk perspective the stock has a very strong current ratio, indicating good short-term liquidity. Value metrics indicate the company has an excellent price earnings ratio, suggesting it may be under-valued.

Which presentation did you choose?

Briefly, why?

3. Both paragraphs contain the *same* sentences. Taking into account **both** the priorities of the investor and the importance of the information, which presentation order will be most effective at helping the investor make a decision?



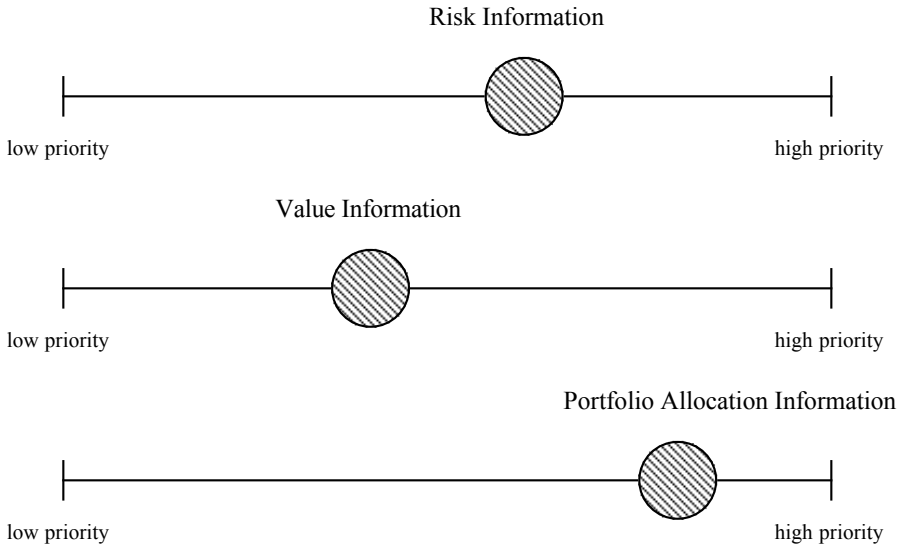
(a) From a value perspective the stock has a history of moderate value metrics. Risk metrics indicate the company has a very poor current ratio, indicating significant potential problems with near-term debt.

(b) Risk metrics indicate the company has a very poor current ratio, indicating significant potential problems with near-term debt. From a value perspective the stock has a history of moderate value metrics.

Which presentation did you choose?

Briefly, why?

4. Both paragraphs contain the *same* sentences. Taking into account **both** the priorities of the investor and the importance of the information, which presentation order will be most effective at helping the investor make a decision?



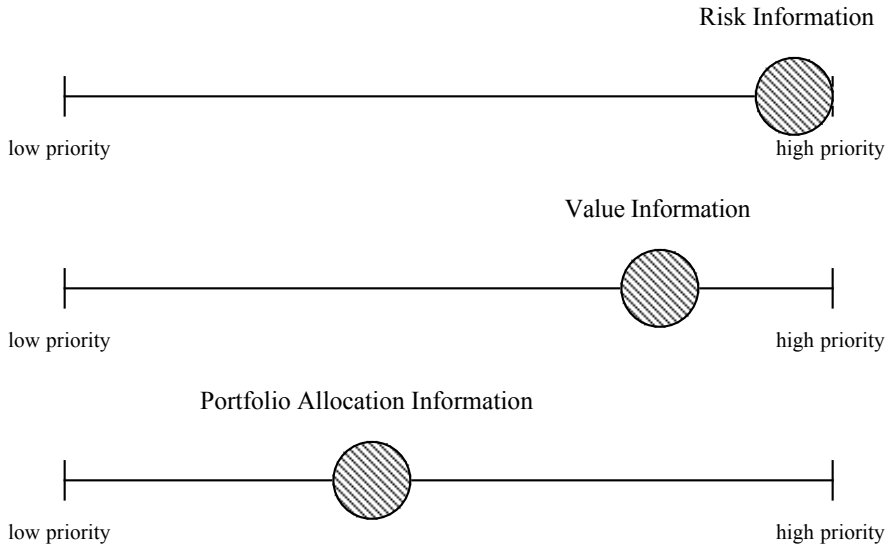
(a) Portfolio metrics indicate this investment is consistent with your stated allocation goal for equities. Risk reports show the company has historically maintained a moderate debt policy. From a value perspective, the stock has a history of moderate value metrics.

(b) From a value perspective, the stock has a history of moderate value metrics. Risk reports show the company has historically maintained a moderate debt policy. Portfolio metrics indicate this investment is consistent with your stated allocation goal for equities.

Which presentation did you choose?

Briefly, why?

5. Both paragraphs contain the *same* sentences. Taking into account **both** the priorities of the investor and the importance of the information, which presentation order will be most effective at helping the investor make a decision?



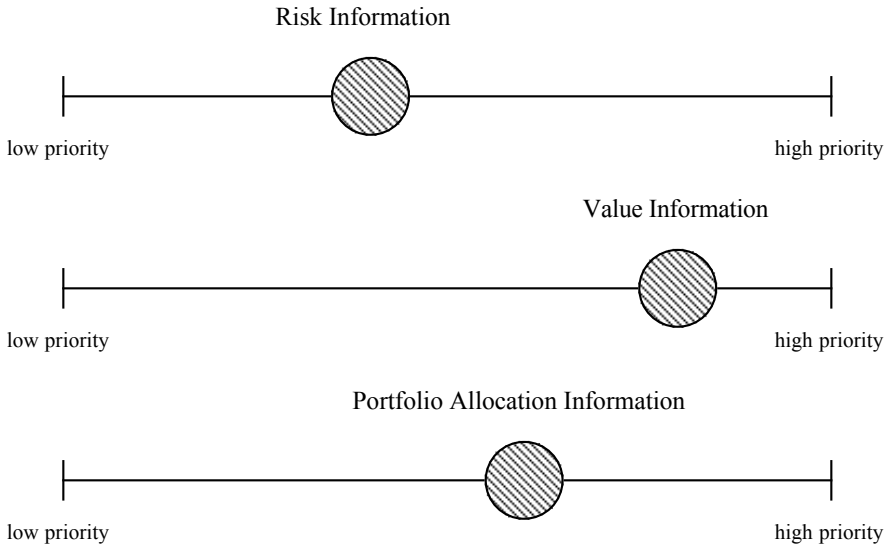
(a) Value metrics indicate the stock has a large negative return on equity, indicating substantial losses. From a risk perspective the company has an acceptable current ratio.

(b) From a risk perspective the company has an acceptable current ratio. Value metrics indicate the stock has a large negative return on equity, indicating substantial losses.

Which presentation did you choose?

Briefly, why?

6. Both paragraphs contain the *same* sentences. Taking into account **both** the priorities of the investor and the importance of the information, which presentation order will be most effective at helping the investor make a decision?



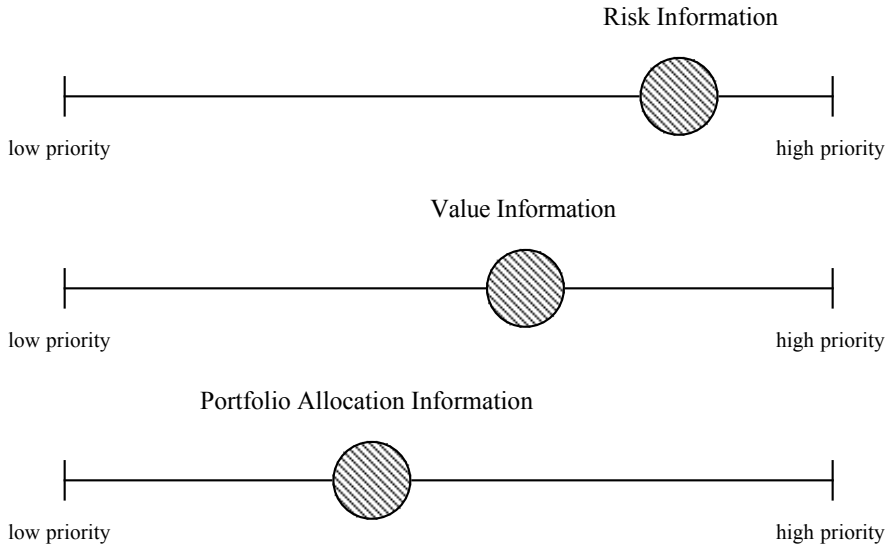
(a) From a risk perspective the company has historically maintained a moderate debt policy. Value metrics indicate the stock has an acceptable price earnings ratio.

(b) Value metrics indicate the stock has an acceptable price earnings ratio. From a risk perspective the company has historically maintained a moderate debt policy.

Which presentation did you choose?

Briefly, why?

7. Both paragraphs contain the *same* sentences. Taking into account **both** the priorities of the investor and the importance of the information, which presentation order will be most effective at helping the investor make a decision?



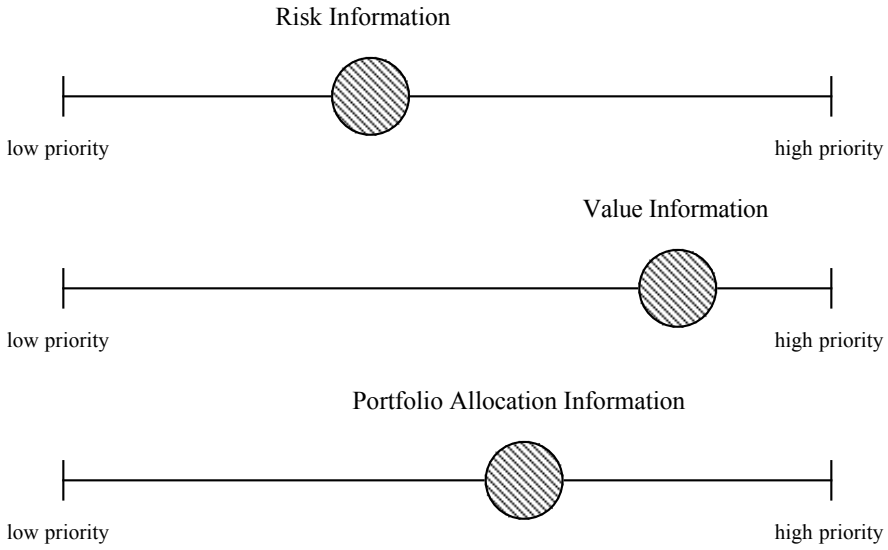
(a) From a risk perspective, the stock's debt equity ratio indicates slightly higher debt than industry average. Value metrics indicate the company has a slightly unattractive price earnings ratio relative to the auto industry.

(b) Value metrics indicate the company has a slightly unattractive price earnings ratio relative to the auto industry. From a risk perspective, the stock's debt equity ratio indicates slightly higher debt than industry average.

Which presentation did you choose?

Briefly, why?

8. Both paragraphs contain the *same* sentences. Taking into account **both** the priorities of the investor and the importance of the information, which presentation order will be most effective at helping the investor make a decision?



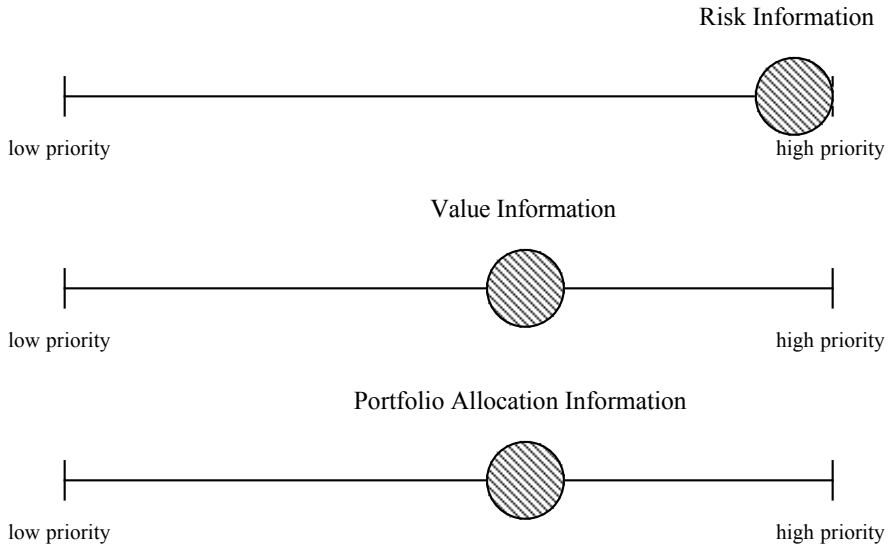
(a) From a portfolio perspective, this investment substantially exceeds your stated allocation goal for equities. Value reports show the stock has a return on equity similar to the industry average. From a risk perspective the company has an acceptable debt equity ratio.

(b) Value reports show the stock has a return on equity similar to the industry average. From a portfolio perspective, this investment substantially exceeds your stated allocation goal for equities. From a risk perspective the company has an acceptable debt equity ratio.

Which presentation did you choose?

Briefly, why?

9. Both paragraphs contain the *same* sentences. Taking into account **both** the priorities of the investor and the importance of the information, which presentation order will be most effective at helping the investor make a decision?



(a) From a risk perspective the company has historically maintained a moderate debt policy. Portfolio metrics indicate you have already exceeded your stated allocation goal for equities.

(b) Portfolio metrics indicate you have already exceeded your stated allocation goal for equities. From a risk perspective the company has historically maintained a moderate debt policy.

Which presentation did you choose?

Briefly, why?

BIBLIOGRAPHY

[AK97] Naveen Ashish and Craig A. Knoblock. Semi-automatic wrapper generation for internet information sources. In *Conference on Cooperative Information Systems*, pages 160–169, 1997.

[App85] Douglas E. Appelt. Planning english referring expressions. *Artificial Intelligence*, 26(1):1–33, 1985.

[ASD05] Farida Aouladomar and Patrick Saint-Dizier. Towards generating instructional texts: an exploration of their rhetorical and argumentative structure. In *Proceedings of the Tenth European Workshop on Natural Language Generation*, Aberdeen, Scotland, August 2005.

[Bal08] W. W. Rouse Ball. *A Short Account of the History of Mathematics*. MacMillan and Co., Limited, St. Martin's Street, London, 4th edition edition, 1908.

[Bar] Barron's online. see http://online.barrons.com.

[BEM01] R. Barzilay, N. Elhadad, and K. McKeown. Sentence ordering in multidocument summarization, 2001.

[Bra02] Steven M. Bragg. *Business Ratios and Formulas: A Comprehensive Guide*. Wiley, 2002.

[CCC94] Jennifer Chu-Carroll and Sandra 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.

[CH97] Sandra Carberry and Terry Harvey. Generating coherent messages in real-time decision support: Exploiting discourse theory for discourse practice. In *Nineteenth Annual Conference of the Cognitive Science Society*, pages 79–84, Palo Alto, California, 1997.

[CHJ02] W. Cohen, M. Hurst, and L. Jensen. A flexible learning system for wrapping tables and lists in html documents, 2002.

[CL97] Charles B. Callaway and James C. Lester. Dynamically improving explanations: A revision-based approach to explanation generation. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, Nagoya, Japan, August 1997.

[Cla98] John R. Clarke. Professor of surgery, drexel university; adjunct professor of computer and information science, university of pennsylvania; and member, committee on data standards for patient safety, board on health care services, institute of medicine, the national academies. Communication during meetings on TraumAid and TraumaTiq, 1997-1998.

[CLR92] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*, chapter 24, pages 504–505. MIT Press, 1992.

[CM99] G. Carenini and J Moore. Tailoring evaluative arguments to users' preferences. In *Seventh International Conference on User Modeling*, Banff, Canada, June 1999.

[CM00] G. Carenini and J Moore. A strategy for generating evaluative arguments. In *International Conference on Natural Language Generation*, pages 47–54, Mitzpe Ramon, Israel, 2000.

[CM01] Giuseppe Carenini and Johanna D. Moore. An empirical study of the influence of user tailoring on evaluative argument effectiveness. In *IJCAI*, pages 1307–1314, 2001.

[CP01] David N. Chin and Asanga 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.

[CS04] Wolfram Conen and Tuomas Sandholm. Anonymous pricing of efficient allocations in combinatorial economies. In *AAMAS*, pages 254–260, 2004.

[DHH97] Chrysanne DiMarco, Graeme Hirst, and Eduard H. Hovy. Generation by selection and repair as a method for adapting text for the individual reader. In *Proceedings of the Workshop on Flexible Hypertext, 8th ACM International Hypertext Conference*, Southampton, 1997.

[DL93] Keith S. Decker and Victor 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.

[dOJB04] Denise de Oliveira, Paulo R. Ferreira Jr., and Ana L. C. Bazzan. A swarm based approach for task allocation in dynamic agents. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, Washington, DC, USA, july 2004. IEEE Computer Society.

[DS97] Keith S. Decker and Katia Sycara. Intelligent adaptive information agents. *Intelligent Information Systems*, 9:239–260, 1997.

[DWS96] K. S. Decker, M. Williamson, and K. Sycara. Matchmaking and brokering [poster]. In *Proceedings of the 2nd Intl. Conf. on Multi-Agent Systems*, 1996.

[EB94] W. Edwards and F. Barron. Smarts and smarter: Improved simple methods for multiattribute utility measurement. In *Organizational Behavior and Human Decision Processes*, volume 60, pages 306–325, 1994.

[ECCC94] Stephanie Elzer, Jennifer Chu-Carroll, and Sandra Carberry. Recognizing and utilizing user preferences in collaborative consultation dialogues. In *Proceedings of UM94*, pages 19–24, 1994.

[fIPA02] Foundation for Intelligent Physical Agents. Fipa communicative act library specification, fipa contract net interaction protocol specification, 2002. http://www.fipa.org/specs/fipa00037/.

[GCL04] Aram Galstyan, Karl Czajkowski, and Kristina Lerman. Resource allocation in the grid using reinforcement learning. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 186–193, Washington, DC, USA, july 2004. IEEE Computer Society.

[GL94] Alan Garvey and Victor Lesser. A survey of research in deliberative real-time artificial intelligence. *The Journal of Real-Time Systems*, 6, 1994.

[Gri75] H.P. Grice. Logic and conversation. *Syntax and semantics*, 3:43–58, 1975.

[Gro77] Barbara J. Grosz. The representation and use of focus in dialogue. Technical Report 151, AI Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, Jul 1977.

[GS86] Barbara J. Grosz and Candace L. Sidner. Attention, intentions, and the structure of discourse. *Computational Linguistics*, 12(3):175–204, 1986.

[GW96] A. Gertner and B. L. Webber. A Bias Towards Relevance: Recognizing Plans Where Goal Minimization Fails. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, Portland, OR, 1996.

[GWC+97] Abigail Gertner, Bonnie Webber, John R. Clarke, Catherine Hayward, Thomas Santora, and D. Wagner. On-line quality assureance in the initial definitive management of multiple trauma: evaluating system potential. *Artificial Intelligence in Medicine*, 9:261–282, 1997.

[HDHP97] Graeme Hirst, Chrysanne DiMarco, Eduard Hovy, and Kimberley Parsons. Authoring and generating health-education documents that are tailored to the needs of the individual patient. In Anthony Jameson, Cécile Paris, and Carlo Tasso, editors, *User Modeling: Proceedings of the Sixth International Conference, UM97*, pages 107–118. Springer Wien New York, Vienna, New York, 1997. Available from http://um.org.

[HG05] Raquel Hervas and Pablo Gervas. An evolutionary approach to referring expression generation and aggregation. In *Proceedings of the Tenth European Workshop on Natural Language Generation*, Aberdeen, Scotland, August 2005.

[Hov88] Eduard H. Hovy. Planning Coherent Multisentential Text. *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*, pages 163–169, 1988.

[Hov91] Eduard Hovy. Approaches to the planning of coherent text. In *Natural Language Generation in Artificial Intelligence and Computational Linguistics*, pages 153–198. Kluwer, 1991.

[Hov93] Eduard H. Hovy. Automated discourse generation using discourse structure relations. *Artificial Intelligence*, 63:341–385, 1993.

[JBV+] Michael Johnston, Srinivas Bangalore, Gunaranjan Vasireddy, Amanda Stent, Patrick Ehlen, Marilyn A. Walker, Steve Whittaker, and Preetam Maloor. Match: An architecture for multimodal dialogue systems, acl 2002 phila pa.

[Jen95a] Nicholas R. Jennings. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artificial Intelligence*, 75(2):195–240, 1995.

[Jen95b] Nicholas R. Jennings. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artificial Intelligence*, 75(2):195–240, 1995.

[JR04] Catholijn Jonker and Valentin Robu. Automated multi-attribute negotiation with efficient use of incomplete preference information. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, Washington, DC, USA, july 2004. IEEE Computer Society.

[KAH94] Craig Knoblock, Yigal Arens, and Chun-Nan Hsu. Cooperating agents for information retrieval. In *Proceedings of the Second International Conference on Cooperative Information Systems*, Toronto, Canada, 1994.

[KD96] Alistair Knott and Robert Dale. Choosing a set of coherence relations for text-generation: a data-driven approach. In Giovanni Adorni and Michael Zock, editors, *Trends in natural language generation: an artificial intelligence perspective*, number 1036, pages 47–67. Springer-Verlag, 1996.

[KF88] Robert Kass and Tim Finin. Modeling the user in natural language systems. *Computational Linguistics*, 14(3):5–22, 1988.

[KL94] Leila Kosseim and Guy Lapalme. Content and rhetorical status selection in instructional texts. In *Proceedings of the Seventh International Workshop on Natural Language Generation*, pages 53–60, Kennebunkport, Maine, USA, June 1994.

[KL95] Leila Kosseim and Guy Lapalme. Choosing rhetorical relations in instructional texts: The case of effects and guidances. In *Proceedings of the Fifth European Workshop on Natural Language Generation*, 1995.

[Klu01] Franco Klusch, Matthias; Zambonelli, editor. *5th International Workshop on Cooperative Information Agents*, Lecture Notes in Computer Science 2182, Modena, Italy, September 2001. Springer.

[KS94] David A. Klein and Edward H. Shortliffe. A framework for explaining decision-theoretic advice. *Artificial Intelligence*, 67(2):201–243, 1994.

[Kus00] Nicholas Kushmerick. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, 118(1-2):15–68, 2000.

[KWD97] Nickolas Kushmerick, Daniel S. Weld, and Robert B. Doorenbos. Wrapper induction for information extraction. In *Intl. Joint Conference on Artificial Intelligence (IJCAI)*, pages 729–737, 1997.

[LHK+98] Victor Lesser, Bryan Horling, Frank Klassner, Anita Raja, Thomas Wagner, and Shelley XQ. Zhang. Big: a resource-bounded information gathering and decision support agent. Technical Report 52, University of Massachusetts Computer Science Department, 1998.

[LHL97] Greg Linden, Steve Hanks, and Neal Lesh. Interactive assessment of user preference models: The Automated Travel Assistant. In Anthony Jameson, Cécile Paris, and Carlo 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.

[LW04] Kevin M. Lochner and Michael P. Wellman. Rule-based specification of auction mechanisms. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, Washington, DC, USA, july 2004. IEEE Computer Society.

[Mar97a] Daniel Marcu. From local to global coherence: a bottom up approach to text planning. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, Providence, RI, July 1997.

[Mar97b] Daniel Marcu. *The Rhetorical Parsing, Summarization, and Generation of Natural Language Texts*. Ph.D. dissertation, University of Toronto, Toronto, Canada, December 1997.

[McC88] Kathleen F. McCoy. Reasoning on a highlighted user model to respond to misconceptions. *Computational Linguistics*, 14(3):52–63, 1988.

[McK85a] Kathleen R. McKeown. *Text Generation*. Cambridge University Press, Cambridge, New York, 1985.

[McK85b] K.R. McKeown. Discourse strategies for generating natural-language text. *Artificial Intelligence*, 27(1):1–41, 1985.

[MD98] Chris Mellish and Robert Dale. Evaluation in the context of natural language generation. In *Computer Speech and Language*, volume 12, pages 349–373, 1998.

[MFLW04] Johanna Moore, Mary Ellen Foster, Oliver Lemon, and Michael White. Generating tailored, comparative descriptions in spoken dialogue, 2004.

[MG96] K. J. Mayberry and R. E. Golden. *For Argument's Sake: A Guide to Writing Effective Arguments*. Harper Collins, College Publisher, 1996.

[Moo95] Johanna D. Moore. *Participating in Explanatory Dialogues*, chapter 3. MIT Press, 1995.

[MOOK98] Chris Mellish, Mick O'Donnell, Jon Oberlander, and Alistair Knott. An architecture for opportunistic text generation. In *Proceedings of the Ninth International Workshop on Natural Language Generation*, Niagra-on-the-Lake, Ontario, Canada, 1998.

[Mor67] Michael S. Scott Morton. *Computer-Driven Visual Display Devices – Their Impact on the Management Decision-Making Process*. Ph.D. dissertation, Harvard Business School, 1967.

[MP90] Johanna Moore and Cecile Paris. Planning text for advisory dialogues. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*, pages 203–211, Vancouver, Canada, 1990.

[MP92] Johanna D. Moore and Martha E. Pollack. A problem for RST: The need for multi-level discourse analysis. *Computational Linguistics*, 18(4):537–544, 1992.

[MP93] Johanna Moore and Cecile Paris. Planning text for advisory dialogues: Capturing intentional and rhetorical information. *Computational Linguistics*, 19(4):651–695, 1993.

[MT83] William C. Mann and Sandra A. Thompson. Relational Propositions in Discourse. Technical Report ISI/RR-83-115, ISI/USC, November 1983.

[MT87] William C. Mann and Sandra A. Thompson. Rhetorical structure theory: A theory of text organization. Technical Report ISI/RS-87-190, ISI/USC, June 1987.

[MWM85] K. R. McKeown, M. Wish, and K. Matthews. Tailoring explanations for the user. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 794–8, Los Angeles CA, August 1985.

[OVDPB01] James Odell, H. Van Dyke Parunak, and B. Bauer. Representing agent interaction protocols in uml. In P. Ciancarini and M. Wooldridge, editors, *Agent-Oriented Software Engineering*, pages 121–124. Springer, Berlin, 2001. http://www.fipa.org/docs/input/f-in-00077/.

[Par88] Cécile L. Paris. Tailoring object descriptions to a user's level of expertise. *Computational Linguistics*, 14(3):64–78, 1988.

[Pet96] Charles J. Petrie. Agent-based engineering, the web, and intelligence. *IEEE Expert*, December 1996.

[Pow03] Dan J. Power. A brief history of decision support systems. World Wide Web, http://DSSResources.COM/history/dsshistory.html, 2003.

[PS04] David Pardoe and Peter Stone. Bidding for customer orders in TAC SCM. In *AAMAS 2004 Workshop on Agent Mediated Electronic Commerce VI: Theories for and Engineering of Distributed Mechanisms and Systems*, July 2004.

[Rei95] Ehud Reiter. NLG vs. Templates. In *Proceedings of the Fifth European Workshop on Natural Language Generation*, pages 95–105, Leiden, The Netherlands, May 1995.

[Rei99] Ehud Reiter. Shallow vs. deep techniques for handling linguistic constraints and optimisations. In *Proceedings of the KI-99 Workshop on May I Speak Freely: Between Templates and Free Choice in Natural Language Generation*, Bonn, Germany, September 1999.

[Ric79a] Elaine Rich. User modeling via stereotypes. In *Cognitive Science*, volume 3, pages 329–354, 1979.

[Ric79b] Elaine Alice Rich. *Building and exploiting user models.* Ph.D. dissertation, Carnegie Mellon University, 1979.

[Rob94] Jacques Robin. *Revision-Based Generation of Natural Language Summaries Providing Historical Background: Corpus-Based Analysis, Design, Implementation and Evaluation*. Ph.D. dissertation, Columbia University, December 1994.

[Saa80] T.L. Saaty. *The analytic hierarchy process.* McGraw-Hill, 1980.

[Sal88] Gerald Salton, editor. *Automatic text processing*, chapter 9. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1988.

[SDB03] Christian Schmitt, Dietmar Dengler, and Mathias Bauer. Multivariate preference models and decision making with the maut machine. In *Proceedings of the ninth international conference on User Modeling June 22-26*, pages 297–302. Springer, 2003.

[SDP+96] K. Sycara, K. S. Decker, A. Pannu, M. Williamson, and D. Zeng. Distributed intelligent agents. *IEEE Expert*, 11(6):36–46, December 1996.

[Sod97] Stephen Soderland. Learning to extract text-based information from the world wide web. In *Knowledge Discovery and Data Mining*, pages 251–254, 1997.

[SP88] Remko Scha and Livia Polanyi. An augmented context free grammar for discourse. In *Proceedings of the 12th conference on Computational linguistics*, pages 573–577, Morristown, NJ, USA, 1988. Association for Computational Linguistics.

[Ste00] Mark Steedman. Information structure and the syntax-phonology interface. *Linguistic Inquiry*, 31(4):649–689, 2000.

[SV00] Peter Stone and Manuela M. Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383, 2000.

[SWWM02] A. Stent, M. Walker, S. Whittaker, and P. Maloor. User tailored generation for spoken dialogue: An experiment. In *Proceedings of Seventh Annual International Conference on Spoken Language Processing*, Denver, Colorado, 2002.

[SZL04] Jiaying Shen, Xiaoqin Zhang, and Victor Lesser. Degree of local cooperation and its implication on global utility. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, Washington, DC, USA, july 2004. IEEE Computer Society.

[WCC+98] Bonnie Webber, Sandra Carberry, John R. Clarke, Abigail Gertner, Terrence Harvey, Ron Rymon, and Richard Washington. Exploiting multiple goals and intentions in decision support for the management of multiple trauma: a review of the traumaid project. *Artificial Intelligence*, 105(1-2):263–293, 1998.

[WGL97] Thomas A. Wagner, Alan J. Garvey, and Victor R. Lesser. Complex Goal Criteria and its Application in Design-to-Criteria Scheduling. *Proceedings of AAAI-97, also a version is available as UMass Computer Science Technical Report 1997-10*, pages 294–301, August 1997.

[WH96] Leo Wanner and Eduard Hovy. The healthdoc sentence planner. In *Proceedings of the International Workshop on Natural Language Generation*, pages 1–10, 1996.

[WJK99] Michael Wooldridge, Nicholas R. Jennings, and David Kinny. A methodology for agent-oriented analysis and design. In Oren Etzioni, Jörg P. Müller, and Jeffrey M. Bradshaw, editors, *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, pages 69–76, Seattle, WA, USA, 1999. ACM Press.

[WMMRS03] M. Wellman, J. MacKie-Mason, D. Reeves, and S. Swaminathan. Exploring bidding strategies for market-based scheduling. In *Fourth ACM Conference on Electronic Commerce*. ACM Press, 2003.

[WP82] Robert H. Jr. Waterman and Thomas J. Peters. *In Search of Excellence: Lessons from Americas Best Run Companies*. Warner Books, New York NY, 1982.

[WW00] 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.

[WWS+04] Marilyn 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.

[WWW98] Peter R. Wurman, Michael P. Wellman, and William E. Walsh. The Michigan Internet AuctionBot: A configurable auction server for human and software agents. In Katia P. Sycara and Michael Wooldridge, editors, *Proceedings of the 2nd International Conference on Autonomous Agents (Agents'98)*, pages 301–308, New York, September 1998. ACM Press.

[YH95] Michael Youssefmir and Bernardo A. Huberman. Resource contention in multiagent systems. In *ICMAS*, pages 398–405, 1995.

[Zac04] Michael H. Zack. The role of dss technology in knowledge management. In *Proceedings of IFIP International Conference on Decision Support Systems (DSS2004): Decision Support in an Uncertain World*, july 2004.

[ZLP05] XiaoQin Zhang, Victor Lesser, and Rodion Podorozhny. Multi-Dimensional, MultiStep Negotiation for Task Allocation in a Cooperative System. *Autonomous Agents and MultiAgent Systems*, 10(1):5–40, January 2005.

[ZM93] Ingrid Zukerman and Richard McConachy. Generating concise discourse that addresses a user's inferences. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, Chambéry, France, aug 1993.

[ZM95] Ingrid Zukerman and Richard McConachy. Generating discourse across several user models: Maximizing belief while avoiding boredom and overload. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1251–1257, 1995.