

Chapter 2

THE RULE-BASED TEXT PLAN INTEGRATOR

2.1 Introduction

This chapter presents RTPI (Rule-based Text Plan Integrator), a system developed in response to problems in the output of the medical decision support system, TraumAID. TraumAID produced messages to help a physician operating in a real trauma care setting. While the messages were individually concise and coherent, they were almost never presented singly. Instead, TraumAID would present a *set* of critiques, and when the critiques shared subject matter they sometimes appeared redundant, disorganized, or incoherent, which would inhibit rapid message assimilation by a trauma physician. The objective of RTPI was to address these problems in the presentation of multiple messages.

RTPI is an implemented system that 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. The resulting integrated text plans must retain and express each of the individual message goals and the means to achieve those goals, while increasing coherence and conciseness relative to the separate messages. Domain-independent rules are used to capture text plan integration strategies, while the facility for addition of domain-dependent rules enables the system to be tuned to the requirements of a particular domain. The system has been tested on a corpus of critiques in the domain of trauma care.

2.2 Introduction

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, Hov91, WH96, ZM95]. However, none have the ability to take a set of independently generated yet inter-related text plans and produce integrated plans that realize all of the communicative goals in a concise and coherent manner.

In contrast, 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.

This chapter describes *RTPI* an implemented system that works with the messages produced by the decision support system, TraumAID. While the examples are taken from the domain of trauma care, the domain-independent rules make the system applicable to other domains of decision support and instruction as well. The motivation and previous work behind *RTPI* is presented in Section 2.3, and

Section 2.4 contrasts it with other work. The heart of the system is *RTPI*'s domain-independent rule base (Section 2.5.2) for integrating text plans. Section 2.6 describes the results of two evaluations: a simple numeric comparison of various message attributes before and after RTPI operates on actual message sets, and a human evaluation of the integrated messages produced by RTPI. The details of RTPI's algorithm are presented in Chapter 3.

2.3 Decision Support in Real Time

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].

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. To accomplish this, TraumaTIQ hypothesizes the physician's plan based on his or her orders and actions¹, and then compares the hypothesized plan for managing patient care with TraumAID's own management plan. TraumaTIQ then identifies significant discrepancies between the plans. 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;

¹ as recorded by a Scribe Nurse, a trauma team member whose purpose is to record information relevant to the case (e.g. the results of a test, or the performance of a procedure). The information is used to answer questions that arise during the case, but also for hospital record-keeping.

3. scheduling errors, which are actions ordered in the absence of orders for pre-conditions 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.

To hypothesize the physician’s plan, TraumaTIQ first chains through TraumaAID’s knowledge/rule base to identify possible explanations for an action; it then evaluates these possible explanations on the basis of relevance to TraumaAID’s current management plan and evidence provided by the physician’s other actions (ordered or performed).

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

Figure 2.1: An example of a single message produced by TraumaTIQ. This message is about an error of omission.

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 TraumaAID’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 TraumaAID’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. An example of a comment produced by TraumaTIQ is seen in Figure 2.1.

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:

1. There informational overlap among the critiques (see Section 2.5.3.1).
2. Some critiques detracted from other ones, or could appear incoherent in the presence of other messages (see Section 2.5.3.2).
3. Some critiques would make more sense if they took explicit account of those appearing earlier (see Section 2.5.4).

Thus a text planner was needed to generate coherent and concise integrated messages that satisfy the multiple goals of the individual critiques.

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. 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 unseen data) revealed 22 common patterns of inter-related critiques, each pattern covering some subset of a critique set. Initially a domain-dependent system, TraumaGEN[CH97], was developed that operated directly on the logical form of the critiques produced by TraumaTIQ.

However, it became clear that many of the patterns were 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 successor to TraumaGEN that uses patterns encoded in domain-independent rules which operate on full representations of RST-style plan trees, so that RTPI is not dependent on the specific logical representation of messages made by TraumaTIQ.

2.4 Related Work

There is currently no other work on assembling or integrating multiple, independently generated text plans when each is intended to function as a component of a larger message. However, there is related work in the areas of text structure, text planning for generation, text plan revision, instruction generation and multi-document summarization.

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 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: they are not explicitly expressed in a clause; they 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; and 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.”

Relations hold between adjacent text spans, and are defined by a list of constraints and effects. Most relations are defined in terms of a nucleus and a satellite, 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.

The contribution of RST most useful to text generation is the idea that recognizing the relations in an RST structure is equivalent to finding the basis for the text's coherence. inter-clause and inter-sentence levels, allowing a single planner to plan both sentences and paragraphs. Hovy [Hov88, Hov91] exploited these properties, constructing text plans that were coherent because they had been built using the principles of RST.

Hovy developed text planning operators 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 is achieved by matching it against the RESULTS field of an operator. The operator posts sub-goals, and the process continues 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 data², since that determined which growth points worked, and by the

² The "available data" was a set of clause-sized, structured information units constructed from a less structured data base using domain-specific rules

choice of growth points associated with an operator. Because the planner selected the text plan that included the highest 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 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 can integrate messages between which there is no recorded 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, such as NV-CAUSE, can be lost when a proposition is related to more than one other proposition in the knowledge base.

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 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[WH96, HDHP97]. It starts with a “master document” and then selects portions and repairs the resulting plan. 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, REVISOR uses a minimalist representation of a discourse plan that removes all but the most essential semantic features for processing. 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 and MADSUM agents both operate 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

minimise 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 system 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].

2.5 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 the goals in the plan. For example, in Figure 2.1 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 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.

2.5.1 Message Integration

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. In particular, the character of the domain influenced how RTPI’s rules handle ordering or re-ordering messages about actions and what the system does with instances of messages that repeat actions.

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.

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 2.2: Original Critiques and a Merged Message

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, change, or cancel an existing order about an action. The organization of the final message affects how well the message supports this purpose.

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 [?].

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. In addition, unnecessary repetition increases message length and sometimes the number of messages.

Reducing message length and message quantity 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 it was clear that reducing the number of words that a physician

had to process (whether read or heard) was an obvious design goal if it could be accomplished without sacrificing clarity. This principle guided rule design towards making both 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 [?].

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 of rules in two ways, as noted in Sections 2.5.3.2 and 2.5.5.2.

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;
5. reflect the system’s social role as a supporter of the physician with ultimate responsibility for the case.

2.5.2 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.

RTPI operates by applying 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. See Chapter 3 for more details on the rules and the algorithm that applies them to a set of messages.

2.5.3 Classes of Rules

RTPI has three 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 we identified from our analysis of TraumaTIQ’s critiques, namely, the need to: 1) aggregate communicative goals to achieve more succinct text plans; 2) resolve apparent conflict among text plans; and 3) exploit the relationships between communicative goals to enhance coherence.

2.5.3.1 Aggregation

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 2.3 illustrates

TraumaTIQ critiques:

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 2.3: Result of communicative goal aggregation.

a set of critiques that involve repetition of suggested actions, and the bottom 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 could 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.

For example, one complex class of aggregation is the integration of text plans that have overlapping actions or objectives, but 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. RTPI produces an integrated text plan comprised of sequentially related segments, with the middle segment conveying the shared actions and their collected motivations. The other

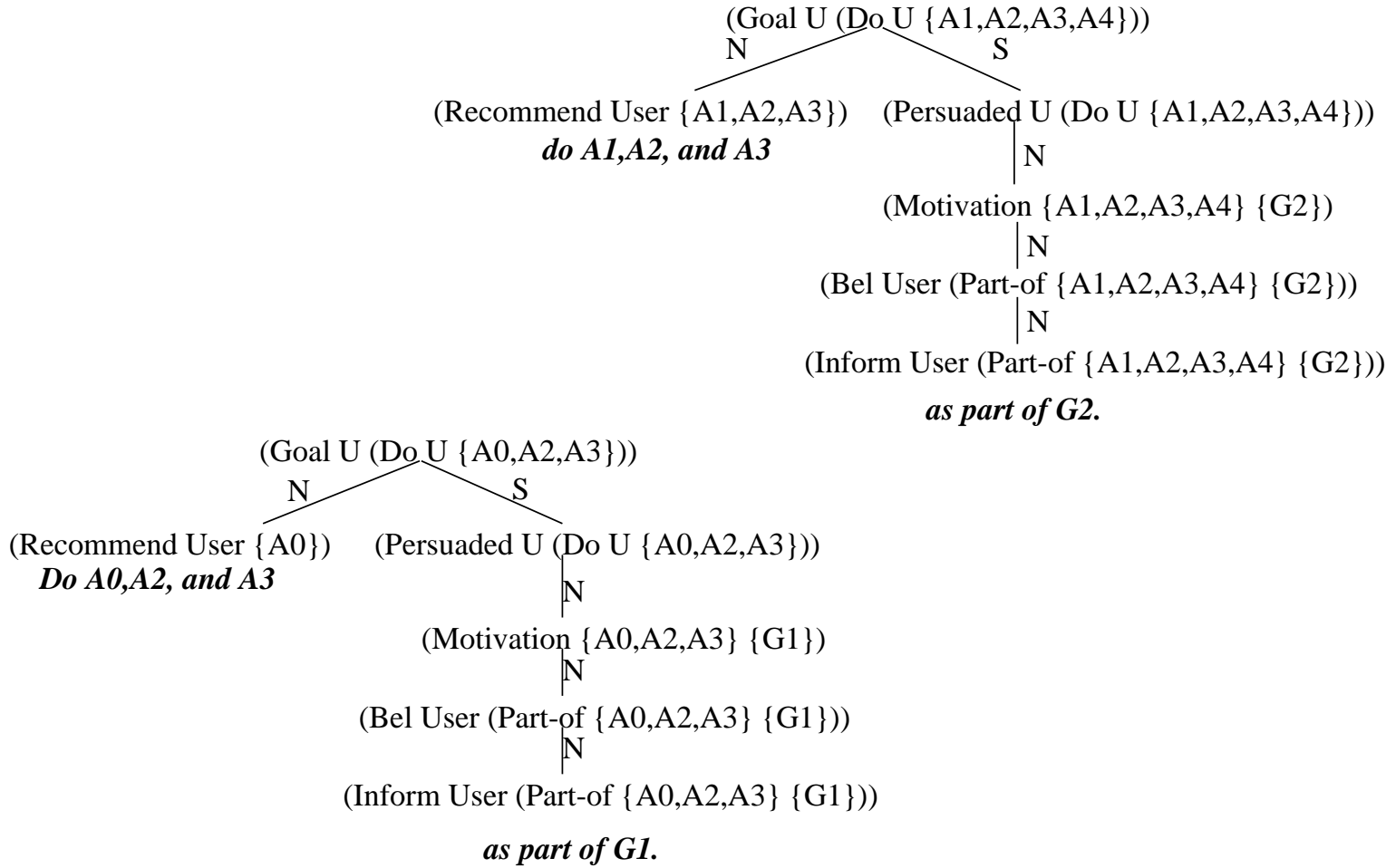


Figure 2.4: Input to *RTPI* (see Figure 2.5).

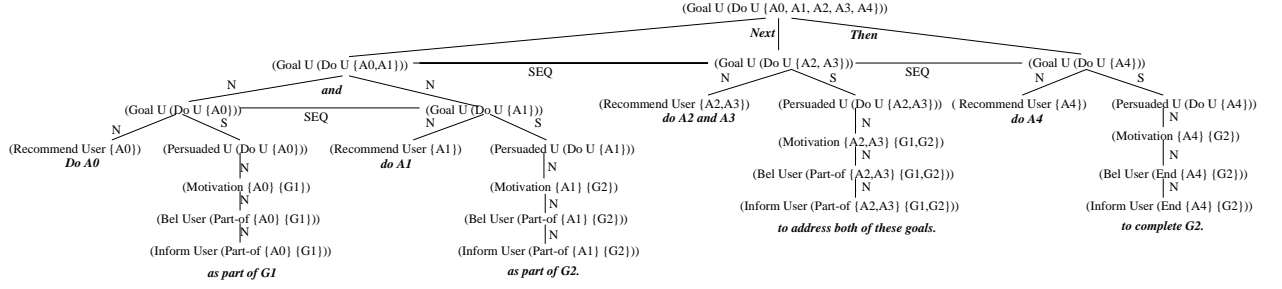


Figure 2.5: Result of a complex aggregation rule (see Figure 2.4).

segments convey the actions that temporally precede or follow the shared actions, and are also presented with their motivations. For example (Figure 2.4), 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 $G2$. Figure 2.5 presents the text plan resulting from the application of this rule. Realization of this text plan in English 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 especially appropriate in a domain (such as trauma care) where the clause re-ordering normally applied to enable aggregation (e.g. Sentence Planner or REVISOR) is restricted by the partial ordering of sequenced instructions.

RTPI can also handle aggregation when actions or objectives are shared between different kinds of communicative goals. The bottom part of Figure 2.3 is the text realized from a text plan that was produced by the application of two rules to three initial text plans: one rule that applies to trees of the same form, and one that applies to two distinct forms. The first rule aggregates the communicative goal (GOAL USER (DO USER check_med_allergies)) that exists in two of the text plans. The second rule looks for overlap between the communicative goal of getting the user

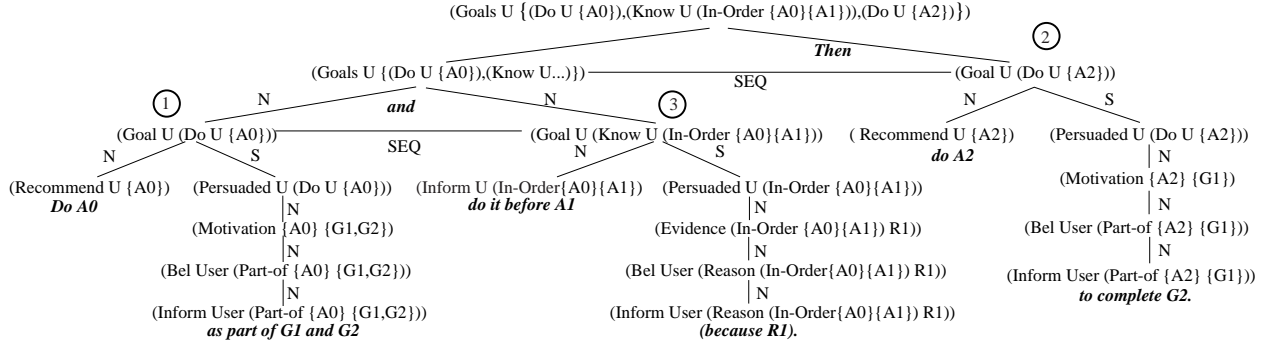


Figure 2.6: Result of two rules applied to input shown in Fig. 2.7. First, a rule that applies to trees with top level goals of the form (GOAL USER (DO ...))uses two trees from Fig. 2.7 to make a tree with the two subtrees labelled (1) and (2). Next, a rule that places scheduling trees ((GOAL U (KNOW U (IN-ORDER ...)))) with related goals inserts a third subtree (3), in this case the entire scheduling tree. A domain specific realizer traverses the tree and inserts cue words and conjunctions based on relations.

to do an action and the goal of having the user recognize a temporal constraint on actions. The application of these two rules to the text plans of the three initial messages shown in the top part of Figure 2.3 creates the integrated text plan shown in Figure 2.6 whose English realization appears in the bottom part of Figure 2.3.

RTPI rules emphasize organization by objectives (appropriate in the trauma decision-support environment). In this application an arbitrary limit of three is placed on the number of sequentially related segments in a multi-part message to facilitate realization, though each segment can still address multiple goals. This allows the reorganization of communicative goals to enable aggregation while maintaining focus on objectives. This limit was sufficient for the input from TraumaTIQ, but may need to be extended when the rule is used for other applications.

2.5.3.2 Resolving Conflict

The ability to recognize and resolve conflict is required in a text planner because *both* the appearance and resolution of conflict can be the result of text structure. *RTPI* identifies and resolves a class of domain-independent conflict, with the resolution strategies dependent upon the social relationship between the user and the system. In addition, the system allows the user to add rules for domain-specific classes of conflict.

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 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 3.2 presents two TraumaTIQ critiques 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 if the parameters indicate the user has more complete knowledge and more expertise.

Clearly the second strategy is most appropriate in TraumAID's domain, where the system is indeed an expert, but with 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 ??.

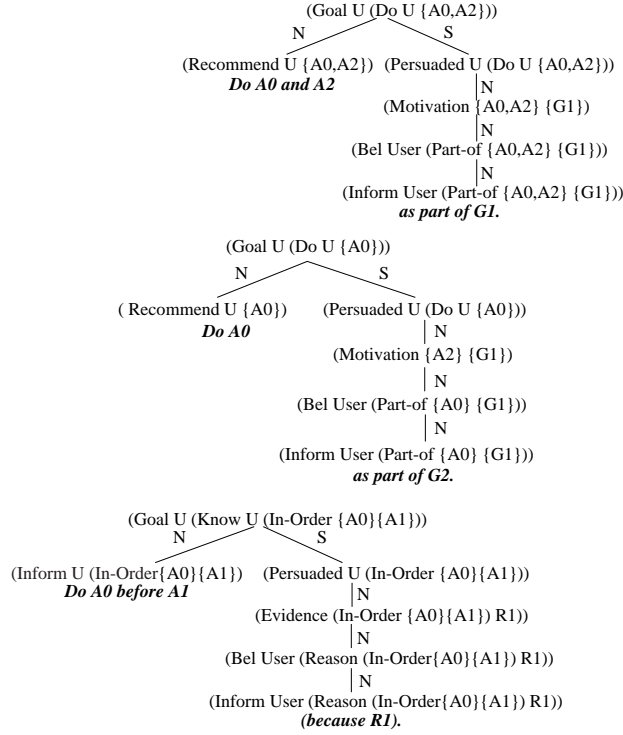


Figure 2.7: Input to *RTPI* (see Figure 2.6).

2.5.3.3 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. For example, consider the two individual critiques produced by TraumaTIQ shown in Figure 2.8. While the two critiques 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, one of *RTPI*'s rules recognizes the 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 dependency between A_s and another action. This rule creates a text plan that results in the message:

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 2.8: Two Inter-Dependent Critiques

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.

2.5.4 Trailing Comments

Occasionally when several text plans (T_1, T_2, \dots) are integrated into a single text plan T_{new} , another text plan T_{out} that overlaps with the integrated plan will remain outside 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 [McK85] 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*.

Unfortunately, when the communicative goal is to get the user to perform an action, trailing comments that refer to such actions have the potential to erroneously

suggest new instances of actions. A solution to this problem is implemented in the text realization templates, where the tree structure:

1. make the focused action the subject of the sentence, reflecting its *given* status in the discourse;
2. utilize clue 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, the main text plan contains the communicative goal of getting the user to perform several actions, including a laparotomy. A SEQUENCE relation is used to adjoin an overlapping text plan as a trailing comment, and this additional communicative goal is realized in English as (clue words underlined):

Moreover, doing the laparotomy is also indicated, along with repairing the left diaphragm, to treat the lacerated left diaphragm.

2.5.5 Other realization features

While most of RTPI’s improvements to a set of messages are due to the integration of text plans into a new structure, the exact wording of the actual text produced from the tree is also important, as demonstrated in the use of cue words in trailing comments (Section 2.5.4). Word selection can thus be used to indicate text structure to a reader. RTPI 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.

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 2.9: Two Dependent Critiques

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. Once RTPI is applied in other domains, the need for full syntactic realization can be re-examined in light of congruencies 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.

2.5.5.1 Focus and ordering

Focus (Grosz, 1977; McKeown, 1985) GET COLLAGEN UPDATED FOCUS REF has been the objective of much discourse research, and it plays several roles in RTPI's generation of messages. As noted earlier, trailing comments capture communicative goals that relate to previously mentioned actions, and cue words are used to shift focus back to the earlier actions. In addition, if there is more than one trailing comment, they appear in order of the most recently introduced action, thus representing successive pops of a focus stack.

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.

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 auto-transfuser 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.

2.5.5.2 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, 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 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, the system would generate 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.

2.6 Evaluation

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 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 critiques 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.³

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 some references to be replaced by pronouns during realization, making the messages shorter and more natural.

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 (shown in Figure ??). 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*

³ 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 critiques appear in sufficient number to consider the effect of *RTPI*.

preferred the messages produced from the integrated text plan 69% of the time.

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. Preliminary evaluation⁴ indicates that *RTPI*'s integrated messages are more comprehensible and a significant improvement over the original critiques. The details of *RTPI*'s operation are presented in Chapter 3.

⁴ A full evaluation must wait for deployment of TraumAID in a live trauma bay, which is not pending.

Chapter 3

RTPI RULES AND ALGORITHM

As noted in Chapter 2

RTPI is designed to take a set of text plans, each with a separate communicative goal, and return a new set of messages which is better suited to the real-time trauma environment.

The approach to this goal was to make the new messages less ambiguous, more concise, and more focused than the original messages whenever possible. It is my intent and assumption that this will make the messages more effective in a trauma setting, though this has not yet been empirically verified (for reasons described in Chapter 2). TrauamTIQ messages which could not be improved remain in their original form.

To make the messages more concise, the system attempts to integrate text plans which share communicative goals. This results in a reduction in both the number of messages and the number of times the names of physician actions and treatment goals are repeated. When combining the text plans, RTPI maintains as much as possible the organization of information according to treatment goals which is present in the original messages.

To make messages clearer and more focused, messages which contain the same information are moved closer to each other, or merged, when possible so that the attention of the physician does not get redirected more often than absolutely necessary. The system also increases clarity by resolving apparent conflicts between messages.

Finally, RTPI attempts to make these changes without forgetting the social role of the system as an expert advisor to the (ultimately responsible) physician.

3.1 Rules in RTPI

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 evaluated. The score function is a heuristic specific to each rule, and is used to determine which rule instantiation 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, 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.

3.1.1 Rule structure

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.

- 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 3.1, replace whatever matched ?b with the result of (aggregate ’and ?b ?d) in ?tree1


```

(scheduling-aggregation
  ;Do {A} before {B} and {D}. <=
  ;Do {A} before {B}.
  ;Do {A} before {D}.
  (;effects
    ((?tree1 . ((?b . (aggregate 'and ?b ?d)))) ;substitutions
    ((tree-remove ?t2 *CURRENT-TREES*)) ;post code
    (length ?a) ;score function
  )

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

```

Figure 3.1: A rule for aggregating temporal constraints.

- other instructions (in Lisp); for Figure 3.1, since the goals of `?tree2` are now incorporated in `?tree1` because of the substitution above, remove `?tree2`.
- score function: In Figure 3.1 the score function is simply the length of the list of actions; intuitively, the rule eliminates one mention of each action in the list, so that is the rule’s degree of goodness.
- List of tree specifications
 - pattern specification: Two trees are listed here, along with a description of nodes that must appear in the tree and conditions about the nodes and their relations to one another (e.g. root, parent, etc.).

- preconditions: predicate statements about variables that must hold for the rule to apply

Figure 3.1 illustrates a very simple example rule. A unifier matches the RST-style tree specification for tree `?tree1` from the pattern specifications of the rule as follows. First, it checks a hash 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 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 3.1, 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`.

If the system decides to apply the rule (i.e. it has the best score), 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`.

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 3.2: An example of a rule removing the appearance of conflict.

3.2 Algorithm

TraumaTIQ does not produce full text plan trees, which are the required input for RTPI. 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 performs rule-based integration of a set of RST-style trees. The order of rules is determined first by rule type and then by score. Rules types are applied in an order designed to maximize derived benefit. The system first applies the rules that resolve conflict, since I hypothesize that the presence of conflict will most seriously hamper assimilation of a message. Next, the rules that exploit relations between text plans are tried because they enhance coherence by explicitly connecting different communicative goals. Then the aggregation rules are applied to improve conciseness. Finally, the rules for trailing comments reduce the number of disconnected message units. Within each rule type group, the rule instantiation with the highest heuristic score is chosen and the rule's operator is applied to the trees using those bindings.

Since the rules are designed to apply incrementally to a set, every application of a rule results in an improvement in the conciseness or coherence 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 one second, even for large (25 plans) input sets, so the time limit is intended to be useful for future, more complex implementations.

When the system has performed as many text plan integrations as possible (or if a preset run time is almost expired) the set of text plans are realized via templates (see Section 2.5.5).

3.3 Integrating Messages

This section reviews some of the implementation detail behind RTPI, and also presents reasons for integration rule design decisions.

An “intersection” is a set of messages which share actions, and so are combined into one larger message with fewer references to the shared actions.

3.3.1 Messages of Omission

Intersections of omission errors are formed when two or more messages about omissions share consecutive actions, subject to the acceptability of the score of the resulting intersection. The benefits and drawbacks of an intersection versus the original messages are weighed in the scoring function associated with each rule.

Omission intersections are limited to three segments (referred to as the *pre-list*, *mid-list*, and *post-list*), each realized as a sentence. This design decision was based on the observation that four or more segments reduced the degree to which the message was organized around treatment goals, making the resulting 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. Consider the following example.

Three original messages, each with its translation:

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

Note that the first two share references to the actionS “check for medication allergies” and “give antibiotics”. The second two messages share references to “do a laparotomy”. These are semantically ordered; medically, doing a laparotomy should not precede either of the other actions, and antibiotics should not be given until after checking for medication allergies. RTPI will usually only refer to these actions in the order they should be performed (unless explicitly giving a message about ordering, shown later) to avoid the possibility of implying an incorrect order.

RTPI’s integrated message:

- Caution: check for medication allergies and give antibiotics to treat the intra-abdominal injury and treat a possible GI tract injury. Then 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 this case, of three possible segments in the first part of the message, only two are used, the *mid-list* and *post-list*. The intersection of actions in the message is for both “check for medication allergies” and “give antibiotics”, since all goals in the

message are related to these actions, and no actions precede them. There is then a *post-list* of the first part of the integrated message which contains actions to be performed after those in the *mid-list*, namely “do a laparotomy”, and explains that is also necessary to address the lacerated left diaphragm.

The third message is then added as a trailing comment (see Section 2.5.4). It was joined to this intersection to maintain focus, because it repeats one of the actions (laparotomy) already mentioned in the intersection. The system notes that this is an additional use for the action by using the word “moreover” or “in addition”.

However, since performing a laparotomy is shared by the goals “treat the intra-abdominal injury” and “treat the lacerated left diaphragm” it is also possible to reduce the number of references to performing a laparotomy. Another way of integrating the same messages would have been:

(not done by RTPI)

- Caution: check for medication allergies and give antibiotics to treat the intra-abdominal injury and treat a possible GI tract injury.
- Next do a laparotomy to complete treating the intra-abdominal injury and to treat the lacerated left diaphragm
- Then repair the left diaphragm to complete treating the lacerated left diaphragm.

This version is not considered by RTPI’s rules because it is not as treatment goal-oriented as RTPI’s version. Even though both messages end up with the same number of total mentions of actions and goals (nine), notice that RTPI’s message has one fewer reference to the treatment goal “treat the lacerated left diaphragm”.

This is because the RTPI version is organizing references to actions around references to goals, in keeping with the strategy noted in Section 2.5.1.

3.3.1.1 Realization of multi-segment plans

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

*****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.

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].

3.3.1.2 Consecutive action sequences

The system is restricted to finding intersections 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 does not appear in the 96 actual test cases,

but it was excluded from consideration for merging after reading the text resulting from hypothetical cases I designed.

3.3.1.3 Different kinds of omission messages

There are three kinds of omission messages produced by TraumaTIQ [?], which are realized as:

Omit1: "Caution: a immediately as part of a."

Omit2: "Caution: a immediately to a."

Omit3: "Caution: check for a to assess the possibility of a."

Omit1 and Omit2 errors can be combined in the same intersection, because they both represent the omission of actions which the system is sure are necessary. Omit3 errors can only combine with each other, since they represent actions designed to rule out a possible condition, not to address a known one. At no time does the TraumaTIQ system produce Omit3 errors which overlap with either of the other kind.

These messages are presented by TramaTIQ at two levels of importance, "WARN" and "INFORM", based on the medical significance of the treatment goal referenced by the message. RTPI considers the levels of messages when applying rules and choosing the order of presentation. Examination of the 96 cases showed that it was not frequent that an action was listed in two messages of different levels, unless one critique was a scheduling type. Upon examining text output it became clear that the benefits of placing INFORM messages in intersections with WARNs outweighed the awkwardness of seeing the same action repeated several messages later. Intersections which contain both WARNs and INFORMs are presented after intersections and messages which are purely WARN. See Section 3.3.1.4 for a more complete explanation of a different result.

3.3.1.4 Trailing Comments

Trailing comments are single messages which share an action with an intersection, and so are joined to an intersection to maintain focus. The objective is to have the text of the critique acknowledge that the action is the same as the one previously mentioned, as would likely happen in natural language. The system denotes that this is an additional use for the action by using the clue-word "moreover". If there is a second trailing comment, it is introduced with the cue phrase "in addition" since it also implies a reference to a previously introduced action.

A trailing comment may need to refer to other actions in addition to the one previously focused on. This is accomplished by subordinating those actions in a phrase introduced by the cue phrase "along with" in a sentence in which the previously focused action is the subject.

The exception to the addition of a clue word occurs if the trailing comment contains more actions than are shared with the intersection; then the trailing comment is moved to the intersection, but no clue word is added *unless* the first action(s) are shared with the intersection.

This example contains a two part "omit" intersection, a simple trailing comment with the clue-word "moreover", a trailing comment with additional actions requiring "along with" cued by "in addition", and a third trailing comment which has no clue word:

- Caution: check for medication allergies and give antibiotics as part of treating the intra-abdominal injury and the compound fracture of the vertebra. Then do a laparotomy to complete treating the intra-abdominal injury, and immobilize the patient and get a neurosurgical consultation to complete treating the compound fracture of the vertebra.
- Moreover, checking for medication allergies and giving antibiotics is also indicated to treat a possible GI tract injury.

- In addition, checking for medication allergies is also indicated, along with getting an IVP, to check for bilateral renal function before laparotomy.
- Caution: pad and position the left leg as appropriate and get the neurosurgical consultation to treat the neuropathy to the left leg.

In the last trailing comment, "neurosurgical consultation" makes it a trailing comment, but "pad and position the left leg" defeats the addition of a clue word. This is because RTPI cannot change the order of actions, due to medical precedence. Thus the critique would have to become: "Pad and position the left leg as appropriate, and also get the neurosurgical consultation to treat the neuropathy to the left leg." There are variations of this phrasing, but because of the ordering restriction, all result in the word "also" apparently (and incorrectly) relating "neurosurgical consultation" to the first action mentioned "pad and position", instead of to the first instance of "neurosurgical consultation" in a previous critique.

Trailing comments are always based on the presence of shared actions between messages, since RTPI messages are organized around treatment goals. This organization prevents multiple messages from sharing the same treatment goal.

Articles in trailing comments are also adjusted to reflect the prior introduction of an action. Note that while "get a neurosurgical consultation" is used in the intersection, in the trailing comment the system uses "get *the* neurosurgical consultation".

Importance levels of trailing comments are also taken into account. A message with level INFORM may be promoted to join an intersection which contains WARNs or mixed WARNs and INFORMs; but a WARN critique cannot be joined to an intersection which contains only INFORMs. This is because intersections are ordered in the output according to the ratio of WARNs to INFORMs, so output is in the order:

1. WARN intersections

2. WARN single messages
3. Mixed intersections by ratio of WARN to INFORM
4. INFORM intersections
5. INFORM messages

Demoting a WARN critique by attaching it as a trailing comment runs the risk of having a physician ignore something important which has been moved down the list; whereas adding an INFORM critique to a higher level intersection achieves the benefit of maintaining focus and reducing the number of messages, does not incur an attention risk similar to that of the demotion, and does not increase the overall number of messages (though it does increase the length of one integrated message).

3.3.1.5 Related text within intersections

If the pre-list or post-list of an intersection has two or more messages in it, the system has rules that recursively look for intersections within that part. These rules are simple clause aggregation, similar to that performed by REVISOR and other systems [?, ?]. Here is an example of a message intersection, first without the aggregation in the post-list section, then with:

Before:

- 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.

RTPI's search for aggregation possibilities (sub-intersections) within intersections is restricted as follows

- Within a three segment intersection, sub-intersections are allowed only if the sub-intersections are single segment (having only a mid-list) intersections. This allows RTPI to use reference phrases (i.e. "both goals") unambiguously in the mid-list of the outer intersection.
- Within a two part intersection sub-intersections are allowed only if they are two part intersections, for similar reasons.
- All sub-intersections must cover the entire pre-list or post-list (no trailing comments will be used in a pre-list or post-list) to avoid distributing actions across too many segments.

3.4 Scheduling Intersections

TraumaTIQ generates six kinds of scheduling messages:

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."

Each scheduling message has two slots for actions, (instead of an action and a goal, like an omission message) which are presented as having a preferred order. TraumaTIQ combines acts which have common goals for all critique types, but does not combine scheduling messages. RTPI does various kinds of combining scheduling messages which share actions.

The first kind of combination is the case shown in Figure 3.1 where the same action appears in the second slot in two of the same type of scheduling critique:

```
action1 before action6
action2 before action6 => action1 and action2 before action6
```

Combinations of this type are simple clause aggregation.

However, even simple combinations of scheduling messages can be improved by careful realization. If two or more scheduling messages of the same type are combined, with a communicative goal of getting the user to recognize several scheduling constraints, then the message is realized with the subordinate clause first to call attention to the ordering constraint. This is done only if the subordinate clause has a single action, 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.

Scheduling messages may combine with omission messages if the actions of the scheduling message are the same as the acts of the omission 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.

becomes:

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

They may also be integrated if the scheduling message contains a subset of the acts of the omission message:

- 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 combine with already integrated omission messages if the arguments of the scheduling message are equal to the arguments of either the pre-list, mid-list, or post-list:

- Caution: get a chest x-ray immediately to rule out a simple right pneumothorax.
- Caution: get a chest x-ray immediately to rule out a simple right hemothorax.
- Do not perform local visual exploration of all abdominal wounds until after getting a chest x-ray. The outcome of the latter may affect the need to do the former.

becomes:

- Caution: get a chest x-ray to rule out a simple right pneumothorax and rule out a simple right hemothorax, and use the results of getting the chest x-ray to decide whether to perform local visual exploration of all abdominal wounds.

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

- omit/scheduling-precondition
- omit/scheduling-urgent
- omit/scheduling-site
- omit/scheduling-precede
- omit/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. Omit-scheduling intersections always share the first action of the scheduling critique, since the second action of a scheduling critique must (by definition) already be in the plan.

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-pri/scheduling-precondition
- scheduling-precede/scheduling-precondition

And scheduling-pri/scheduling-site is handled but doesn't occur. Known combinations which cannot logically occur are intersections between the first action of the following scheduling messages:

- scheduling-urgent/scheduling-pri
- scheduling-urgent/scheduling-precede
- scheduling-urgent/scheduling-site
- scheduling-urgent/scheduling-dependency
- scheduling-pri/scheduling-site
- scheduling-pri/scheduling-dependency
- scheduling-precede/scheduling-site
- scheduling-precede/scheduling-dependency

Also, overlap between the first action of one scheduling critique and second action of any other scheduling critique cannot occur since that implies a three-way ordering restriction; for the restriction to be detected, one action from each critique must be in the physician plan, and so of three actions, two are in the physician plan. If two are in the physician plan, then one critique would not have been generated, since both actions of one critique are in the physician plan.

3.5 Removing Apparent Conflict

This section elaborates on Section 2.5.3.2. There are certain combinations of 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.

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.”

one pretty, blonde princess exists in the knowledge base. However, RTPI is unique in addressing apparent conflict that is introduced by when messages from two coherent sources are juxtaposed.

Only scheduling-preconditions with errors of commission, or scheduling-preconditions with procedure-choice errors create this apparent conflict. 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.

Of these three combinations:

- proc-choice1/scheduling-precondition
- proc-choice2/scheduling-precondition
- proc-choice3/scheduling-precondition

only proc-choice2/scheduling-precondition occurs in the actual trauma test cases, but all three are handled by the system. 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; these did not occur in the test cases.)

3.6 Summary

rules

RTPI’s rules are designed to work with RST-style trees. RTPI is written in Lisp, but rules can be added to the system without programming, simply by following the rule template in this chapter, and the examples in Appendix ??.

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. RTPI's rules were designed to handle general cases of the kinds of messages produced by TraumaTIQ, not simply the exact messages here. The specific sub-categories of messages and the kinds of integration performed between them 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. Within each rule set, a score function associated with each rule determines which rule is applied next. After each rule application the text plans are a viable set of messages; improvement continues until no rules are applicable.

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

The previous chapters focused on the problem of generating effective messages in real-time decision support. I presented my solution, TraumaGEN, 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. TraumaGEN 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, 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 delivered after the close of the market).

The situation is complicated when the decision support environment is not static. In addition to user priorities changing, a user's status may change, or the environment could change in a way that affects a user's relative status. For example, a user's ownership of an asset such as land may be static, but a flood or a change in zoning laws may change the user's status by affecting the value of the land.

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

The remainder of this chapter outlines my solution: 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 poor 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 and tested in the system MADSUM (Multi-Attribute Decision Support via User Modeling).

4.1 An 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. The 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 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.). Furthermore, the use of an appropriately designed agent architecture allows the system to dynamically respond to changes in the environment and/or user priorities.

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, and 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 source and task 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, ?]. 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.

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 user preferences of 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 5.5.2.2) so that the wrapper agents can produce text plan trees for the task agents above (and eventually the Presentation Agent) to integrate.

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 source and expert agents, since there exists an accepted body of standard methods of numeric analysis. (Contrast this with decision support for buying art, or adopting a child.)
- It is a domain that 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 allow 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. 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. Most user interactions will involve only the second part.

The “Basic” screen of the interface is the heart of the user-MADSUM interaction. Figure 4.1 shows the graphical sliders employed by the user to indicate relative priorities for the attributes in the utility function. The user can also enter 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 message¹ which is sent to the MADSUM Presentation Agent to begin the process. At the conclusion of the process a new screen shows MADSUM’s resulting text message.

More advanced features of the MADSUM interface are accessed via the second screen (Figure 4.2). Here the user can modify the separate functions that control how utility is derived from each attribute term (see section 5.3.4). 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. A decision support system must represent these differing priorities and related constraints in a user model, and then use that model to allocate resources for an unseen

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

task across multiple agents in a dynamic environment. MADSUM 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 organization responds appropriately and predictably in the presence of inevitable information failures.

BasicAdvanced

MADSUM Message Sender

Use the sliders to rate the importance of each kind of information:

Risk

Value

Portfolio Goals

Use the sliders to rate the importance of these message attributes:

Text Length

Message Cost

Choose a stock:

HPQ

IBM

INTC

IP

shares

100

sharePrice

82.75

Target value	Optional Constraints	
	min	max
50	0	70
5	0	10

Set these attributes

Send New Job Criteria

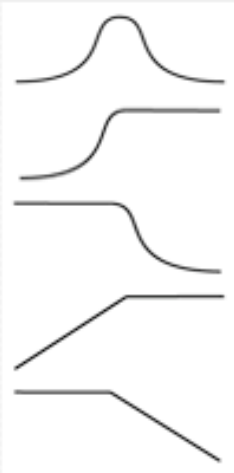
Change Job Criteria

Figure 4.1: The basic MADSUM interface for user input.

Basic

Advanced

Select function



Attribute Name

time

Behavior of value in function:

value:

change point (end of plateau)

Optional Attribute Constraints

min

max

Term Coefficient

0

120

SET these term features

See FIPA message

Figure 4.2: The advanced MADSUM interface for user input.

Chapter 5

A DECISION-THEORETIC APPROACH TO RESPONSE GENERATION

5.1 Introduction

As discussed in the previous chapter, an effective decision-support system must produce responses that are tailored to the individual user. In producing such responses, the system must not only take into account a user's preference 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 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 repeatedly inferred. One of several information hierarchies is selected based on the detected topic of the last question and the current *relevant* user goal, and the different contents of the hierarchies trigger different rules, which determine the output.

Paris [Par88] modeled a user’s level of expertise, and varied a response not only by modifying the content, but also by modifying presentation style. MADSUM does not make any high level decisions about style of presentation, 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 used a detailed, pre-existing user model to design a response when the system detects that the user has a misconception. This work was the first to address the “overlay” assumption issue, i.e. to consider the possibility that the user’s knowledge/beliefs are not a subset of the system’s knowledge, but may simply intersect (or, as in RTPI, may be a superset).

All of these early systems modified the system response to a user 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).

5.2.2 Modeling User Preferences

A very early system using a model of user preferences, GRUNDY [Ric79] employed stereotypes to instantiate individual user models to make book recommendations. This strategy reduced the number of questions that the system had to ask

the user before it had a complete enough user model to recommend a book. Individual facts, such as gender, would expand via stereotype to imply preferences for or against books that were romantic, violent, intellectual, etc. These preferences would be assumed correct until the system had contrary information (of a stronger nature: the system also kept track of why it believed what it did) about the individual's preferences, at which point the model would be updated.

Elzer et al [ECCC94] focused on identifying user preferences 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 determining an action plan that better fit those preferences than the user's original plan.

A more recent effort at acquiring user preferences re-examined the problem of asking the user questions to determine preferences. Chin and Porage [CP01] use multi-attribute utility theory to make travel choices for a user. At each step in the decision process, their system chooses the next question based on maximum information gain (i.e. the best reduction in utility uncertainty). Work of this kind may eventually facilitate the use of systems like MADSUM that require user preferences to operate.

5.2.3 Tailored Response Generation

Adaptive systems have used concepts of utility theory, either informally or formally, to make decisions that take into account the user's preferences. 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.

More recently, researchers have begun to design their responses to incorporate user preferences by using formal multi-attribute utility functions.

MATCH [WWS⁺04, JBV⁺, SWWM02] consists of a sophisticated user interface on specialized hardware for restaurant choice based on user preferences expressed in 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 cost domain, it is not suitable for attributes in other domains. For example, MADSUM allows a user to express that increased length is good up to a point, then bad beyond that point.
2. MATCH uses ordinal assignment of weights, versus the arbitrary weights in MADSUM. MATCH's SMARTER [EB94] procedure for eliciting multi-attribute decision models cannot express e.g. that risk and value information are both of equal high value. 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.
3. The MATCH utility function computes the utility of each *restaurant* to the user, not the utility of the system's *message* to the user. MATCH is principally an argumentative system, making a choice and attempting to convince the user that the choice is correct. MATCH may choose to leave important counter-arguments out of its presentation if those arguments are not considered part of the most persuasive message. This is appropriate when the user's knowledge

and expertise are a subset of the system's, but not when the user has final responsibility for making a critical choice.

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. In particular, crucial parts of MATCH appear to be “distributed”, e.g. the text planning module.

FLIGHTS ([MFLW04]) uses a multi-attribute utility function in a user-model to influence the generation spoken language of airline flight recommendations. 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 available.

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 explicitly includes user influence on attributes of the system's output, in terms of cost, length, and time, as well as domain-specific preferences such as topic. Thus MADSUM is concerned with the total utility of the *message* to the user, including the utility of the information provided as well as 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, since 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

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 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` and dollar `COST`, 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). Although User Attributes could be captured in a long-term user model that is constructed and revised over time, the Constraints and the Utility Function will vary with different user interactions and perhaps even change during an interaction.

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 ([Ric79]) 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 affect the significance of certain pieces of information (see 5.3.4). 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. Soft constraints must be implemented as part of the utility function to operate in this manner, and are further discussed in 5.4.0.1.

5.3.4 User Preferences

User preferences (that are not hard constraints) influence MADSUM output via the utility function. A user expresses preferences about one or more attributes of the output. MADSUM is designed to accept an arbitrary number of attribute preferences, but for my implementation I selected the attributes of text length (LENGTH) and dollar cost (COST), and three domain specific attributes, one for each of three topics in the financial investment domain (see 4.2). These topics are RISK, VALUE,

and GOAL. With these the user can indicate his or her preference for information on a certain topic within the broader category of information about the investment under consideration. 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 (see 4.3).

The slider for each preference covers a one to ten continuum. The output of the slider, as a real number, is included in a message to the Presentation agent (see 5.6) that starts the decision support process. Each message attribute is represented by a term in the user's utility function, and MADSUM uses the user preference number from the slider as a coefficient for that attribute term (see 5.4). The rest of the term consists of the actual attribute value and a function that maps it into a 0..1 valued space, discussed in 5.4.0.1.

The attribute value is either a hard number, as for LENGTH and COST, or a number representing information significance for a topic preference (see 5.4). Attributes can capture characteristics of propositions that might be presented to the user, and for information significance the value captures the specificity of a set of propositions to the decision task at hand — i.e. its significance 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 both its absolute debt-to-equity ratio (heavy debt is bad) and the particular industry (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. For example, suppose a source agent, DebtEquity, obtains the debt-to-equity ratio for company XYZ, 0.6, as well as the industry average debt-to-equity ratio of 0.8, and sends it to the task agent (its wrapper) DebtRisk. DebtRisk uses a domain-specific and ratio-specific formula¹ to calculate how good or bad XYZ's ratio is in light of the industry ratio. In this case, a value of 0.6 means that for a company of its size² 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 (see 5.5 for information on how DS is used in assembling text plans).

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 a proposition that addresses the appropriateness of the investment from an age perspective may depend on how close the user is to retirement. In my system, I currently compute the DS of a set of propositions in a particular category (RISK, VALUE, or GOAL) as the sum of the DS values

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 []

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

for the individual propositions; this has worked well in my application but further experimentation is needed to fully validate the decision.

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. Within each triplet the first number represents the single highest DS part of 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

The MADSUM system is subject to misunderstandings if the user is not familiar with the concept of relative preferences, as found in utility functions. 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

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.

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 .

$$Utility = \sum_{i=1}^n w_i f_i(a_{value_i}) \quad (5.1)$$

The weights w_i , giving the importance of each attribute to the user, are extracted from the positions of sliders that are manipulated by the user in a graphical user interface. For resource attributes such as length of response or processing time, a_{value_i} is the actual value of the attribute, but as noted previously, for topic DS attributes a_{value_i} captures a numeric representation of significance to the decision at hand.

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

Figure 4.2 illustrates the currently implemented functions in MADSUM's predefined library of utility functions. Each function can also take an additional argument (which the user can set in the advanced features section of the user interface); the effect of the additional argument varies according to the function, and is noted in the function descriptions below:

1. f_{Normal} approximates utility as a normal distribution of the possible values of its parameter. The additional parameter specifies the center of the distribution

and the spread.

2. $f_{StartPlateauNorm}$ 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.
3. $f_{EndPlateauNorm}$ is same as above, except that the curve rises to a plateau.
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.
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 $x = 2 * changePoint$.

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

5.4.0.1 Soft Constraints

The user specifies soft constraints by using the appropriate f_i and then using the additional argument to determine its shape. The user interface shows how the additional argument relates to the f_i in question. For example, 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 $1/2$ 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.

MADSUM has an easily extendable library of base utility functions. For example, 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 length of the response until the soft limit is reached and then decreases; such a user might select the base function f_{Normal} to evaluate the contribution of length to the overall utility of the response.

Thus the magnitude of a term in the overall utility function is affected by the value of the attribute in the environment (either its actual value in the case of resource attributes or the DS value computed from propositions in the case of information attributes), the base utility function f_i 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). 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 RTP1 described in Section ?? 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).

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 a single program 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. Unlike MADSUM, ILEX is not designed to take an arbitrary set of other result attributes; nor does ILEX use a utility function to control attribute trade-offs in a decision-theoretic manner designed to satisfy user preferences.

Marcu's bottom-up approach (also a standalone system) was the first to be designed to express all of the content chosen for expression. 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 four simple rules for combining pairs of text plan trees. The system is designed to operate with additional rules, if

necessary, and different rules can be turned on and off with ease.

5.5.2.1 Integration rules

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 bound to design a text plan tree incorporating all of the received component trees. The set is first ordered by an altered version of the utility function. Since resources such as overall text length, cost, and time to produce 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, since they apply to a whole, the altered utility function excludes the resources length, cost, and time, and is calculated only on DS. Individual rules place components according to peak DS, with the highest peak DS 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. Since rules are applied to all combinations of trees, from the bottom up, the resulting tree and text will have the highest DS text fragment first.

This seems counter-intuitive at first; the component trees were chosen based on the total utility they provide, 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 suspected 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 to by limited testing (see 5.4) and should be further explored.

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 that 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 RISKagent, 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. They show sub-optimal results for two kinds of ordering (majority rule⁴ and temporal) but the best result when they use information *theme* (similar to topic) to group sentences.

The first 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 the 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, so that the set consists of both small component trees and partially or fully assembled trees.

1. The Elaboration rule (Figure 5.1) 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 orientation, and if one supporting clause is substantially more significant (higher DS) than the other.
2. The Joint-Under rule works similarly, but is used when the two supporting clauses are of the same or similar DS. This is also a form of aggregation.

⁴ recall that sources are producing duplicate, or almost duplicate information

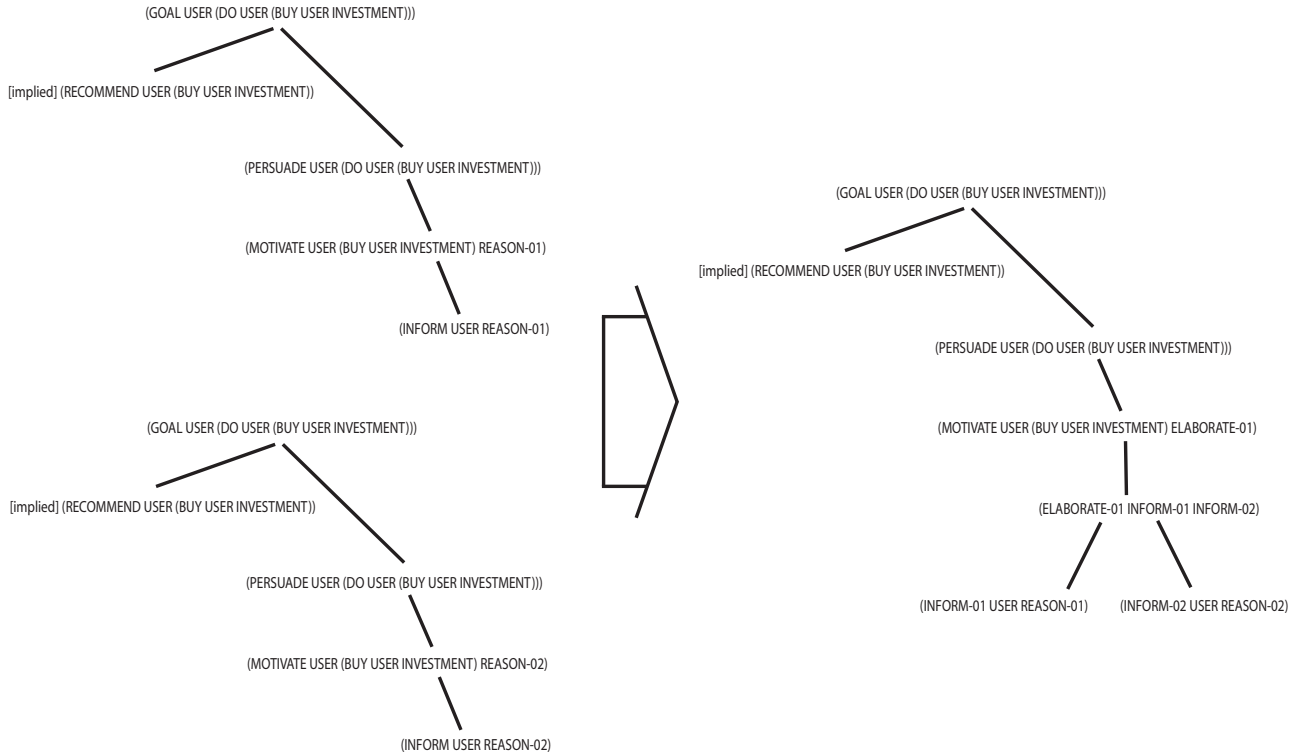


Figure 5.1: Two simple text plans, and the result of applying the Elaboration rule.

3. 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 ??).
4. Finally, when two trees cannot be joined any other way, a Joint relation can be superimposed over them. This is avoided if possible since a Joint contains no useful semantic content to aid realization (or reader understanding).

5.5.2.2 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 Elaboration 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]. 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 possible 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.

The agent architecture (see Chapter 6) uses a formal agent communication language to transfer information between agents that may be operating on different machines. This involves a high degree of communication overhead. Also, the system has many features designed to make it robust, and these add further communication requirements.

The system takes about 17 seconds for the two full rounds of communication and simple 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. Those figures can be reduced substantially when the parallel nature of the architecture is leveraged, and when full Java runtime optimization is turned on (it has been turned off for MADSUM timing runs since the optimization effects are cumulative over a series of runs, rendering consecutive runs incomparable). For more 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 5.2 displays MADSUM's response under different soft constraint and priority settings. In Figure 5.2a, 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 the responses in Figure 5.2b and Figure 5.2c, 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 5.2a and 5.2b, but were altered in Figure 5.2c 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 5.2b, 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 5.2c, 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.

5.2a: Risk metrics indicate IBM has a low debt-equity ratio, suggesting the ability to weather an economic downturn; further, the company has a strong current ratio, indicating good short-term liquidity. In addition, IBM has historically maintained a moderate debt policy, and the stock has maintained a moderate risk profile. On the other hand, from a portfolio perspective you have already exceeded your allocation goal for equities. Value metrics indicate IBM has a price earnings ratio similar to the tech industry average.

5.2b: Risk metrics indicate IBM has a low debt-equity ratio, suggesting the ability to weather an economic downturn; further, the company has a strong current ratio, indicating good short-term liquidity. On the other hand, from a portfolio perspective you have already exceeded your allocation goal for equities.

5.2c: Value metrics indicate the stock has a price earnings ratio similar to the tech industry average; on the other hand, from a portfolio perspective you have already exceeded your allocation goal for equities.

Figure 5.2: Three responses, derived from different soft constraints and priority settings.

5.8 Evaluation

I have implemented and tested the MADSUM architecture for adaptive response generation in a financial investment domain. The examples in Figure 5.2 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.

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 (see 5.5.2.2).

5.8.1 Selection

To test the system’s ability to select appropriate content, I performed an experiment in which 21 subjects were each presented with 7 scenarios (see Appendix ??). Each scenario consisted of 1) a graphic depiction of sliders representing user priorities for the three kinds of content (RISK, VALUE, and GOAL) 2) two sets of propositions, one of which had been produced by the system. The alternative was hand generated using the strategy *not* used by the system, i.e. if the system response was ordered by user preference, then the alternative presented was ordered by significance (DS). Sometimes the system’s response was listed first and other times the alternative appeared first. The subjects were asked to determine, given the user’s slider settings for content priority, which content set was most appropriate for that user. In some scenarios, the significance (DS value) of propositions and the priority

that the user placed on that kind of information were congruent (each proposition was either both significant and high priority, or both of lesser significance and lesser priority), and in other scenarios the significance and priority of propositions conflicted. The alternatives to the system's responses were constructed to give subjects the opportunity to choose between responses that favored priority in content selection, responses that favored significance, and responses that balanced priority and significance as is done by MADSUM's utility function. I applied a one-tailed binomial test to the results which showed that the subjects had a statistically significant ($p < .01$) preference for MADSUM's strategy of balancing significance and priority in content selection.

5.8.2 Ordering

As discussed in Section 5.5.2, MADSUM places the highest utility propositions early in the response, subject to coherence constraints on the construction of text plan trees. To test the system's ordering of propositions in its presentation to the user, 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. But in this experiment, the subjects were presented with two different orders of presentation of the same propositions. In each case, most cue phrases and connectives were removed in an attempt to prevent subjects from being influenced by phrasing. Once again, a one-tailed binomial test showed statistically significant ($p < .01$) support for the system's decisions about order of presentation of propositions. This was true even when proposition significance (DS value) and user priority for that kind of information conflicted, although the level of statistical significance for this subset of the scenarios dropped to ($p < .05$).

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, our evaluation experiments support the content selection and ordering strategies used by MADSUM and suggest that they will contribute to 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 time to produce. 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.

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
2. how to allocate finite resources across sources
3. how to integrate results from different sources
4. what to do when sources don't perform as agreed or results do not meet expectations
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 6.3 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. MADSUM presents new ideas in each of these areas, which are noted in the context of previous work.

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 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 DECAF, while BIG implements it and is investigating design issues. In particular, BIG reasons about uncertainty, and opportunistically plans to address uncertainty. However, that planner is distinct from the scheduler, which is not immediately concerned with uncertainty about information, but rather the end-to-end performance of the system. The negotiation process that will determine how conflicts between these two are resolved is a research topic of great interest to information agents, but not directly to MADSUM, which incorporates information agents as sources. In fact, BIG would make a terrific source agent for a MADSUM implementation, since the forms it creates with results would easily map to RST-style templates.

As in MADSUM, the user can also place constraints on the amount of time and money BIG will spend in developing an answer. BIG does allow the user to specify one preference about the information result it produces. The user can indicate a preference for information precision versus breadth of coverage, 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 preference number, 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, while BIG uses a single preference 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 [WWS⁺04] and [MFLW04], as reported in Section 5.2.3, namely that BIG builds a model of each product that it discovers, based on a list of user-specified features, 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 5.2.3, MADSUM pays attention to the utility of the *message* to the user, not the utility of a model outcome.

6.1.0.1 Architectures that facilitate utility theory implementation

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 minimizes the effort required from the user by

employing graphical sliders, but if a system had a very large number of utility function terms this would become unwieldy.

The MAUT Machine [SDB03] (for **M**ulti-**A**tttribute **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 associated with each term (see 5.4), and then to “mark up” the products in the catalog with the information required to calculate an item’s utility to the user.

¹ 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.

Another method of 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. In each consecutive step in the decision process, their system chooses the next question based on a calculation of maximum information gain (in this case, the best reduction in utility uncertainty). 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 next query to the user.

¹ Again, this design as currently implemented does not calculate the utility of the information provided to the user, but rather the utility of an object. I see no reason, however, that this system could not be implemented as part of a system that measures the utility of an information result, provided that it operates quickly enough to enable the comparison of the utility of hundreds or thousands of possible text plan trees.

These two systems implement technologies that could enhance the user experience of MADSUM by facilitating the user’s expression of preferences. Making this part of the system more attractive to the user could enable the elicitation of more preferences, leading to results that better satisfy the user.

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] 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 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](e.g. submitting only one bid in a progressive auction, instead of a basket). 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.

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 choose the correct 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 cooperative 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

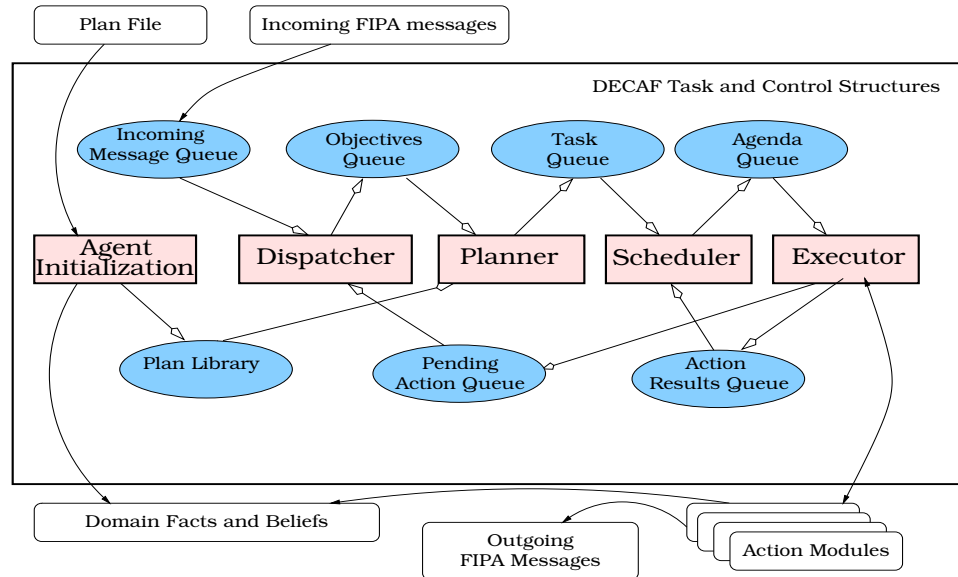


Figure 6.1: DECAF Architecture Overview

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 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]). Functionally, DECAF is based on RETSINA [SDP⁺96] and TÆMS [DL93].

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 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 (rounded 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.

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. Third, my original plans for the MADSUM system included taking advantage of DECAF’s unique ability to make scheduler choices at runtime. This feature of MADSUM is now intended for future implementation. 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, currently consisting of thirteen DECAF software agents. At the lowest level are seven information providers that can access information, sometimes from remote sources. The implemented providers are

1. Current Ratio: this agent returns the eponymous financial ratio for a given company. (The Current Ratio is an indication of the ability of a company to cover short-term obligations²). For a higher price the agent will report on this company’s current ratio relative the that of the company’s primary industry classification (e.g. DCX (Daimler-Chrysler) industry classification is auto manufacturing).
2. Debt-to-Equity: a financial ratio agent, see Current Ratio. This ratio is an indication of how highly leveraged a given company is.
3. Industry Risk³: 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 [?].

² 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].

³ 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.

4. Price-Earnings: a financial ratio agent, see Current Ratio. 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, see Current Ratio. 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, users enter their 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 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 following agents are wrappers, converting the information from the 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.)

1. Debt Risk: parent to Current Ratio and Debt-to-Equity, this agent handles source agents that report information on debt-related aspects of a company's financial picture.
2. IR Parent: This agent is solely a wrapper for Industry Risk (see footnote for same).
3. Goal: parent to Portfolio Goal and Retirement Goal, this agent wraps sources that consider the user's explicit or implied financial goals.
4. 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 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 results from agents below fails to meet expectations (see Section 6.7). 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.

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 6.2). In the simplest version of the protocol an agent *A* requests that agent *B* make a proposal (bid)

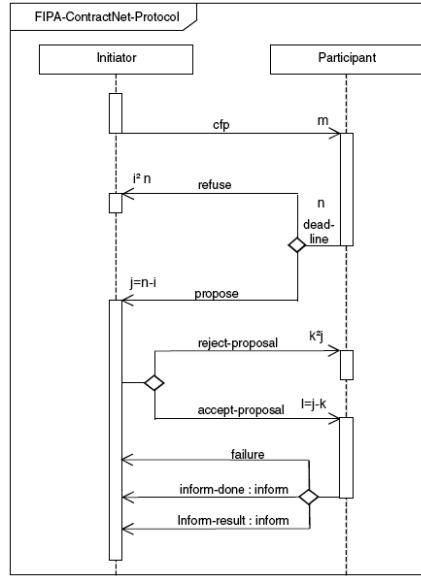


Figure 6.2: FIPA Contract Net Interaction Protocol Specification[FIPA02, OVDPB01]

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 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, 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 all agents 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 5.3.4) 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 that have more than one level of service 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 6.7) 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 parameter space (this provides options during later parts of the task). At each level agents will consider various combinations of the bids submitted in terms of utility, determine a “basket” of combinations, and propagate new bids representing the basket up the tree. This occurs recursively to the top of the tree. Though agents submit multiple bids, each combination includes at most one contribution from each child agent; thus the complexity of the problem may be constrained by limiting the number of children of an agent. More precisely, since each basket of bids considered has at most one bid from any given child, then for n children and a maximum number of m bids per child, the number of combinations considered by a task agent is

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

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* relation in TÆMS[DL93].

6.4.2.1 Alternative bids

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 based on utility, then *all* might well be unattractive. Bid diversity will also likely be valuable under certain failure circumstances, but that is not demonstrated here.

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, when in fact it may only be orthogonal. 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 apriori 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 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 until the desired number of alternatives (plus one) is reached (the system default is three alternatives; the plus one is for the tree that will contain the high bid). 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 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.

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 6.3.1).

P must now determine what bids to send to her parent. P considers all possible combinations, in this case fifteen. If P chooses to submit the six highest utility bids, then the six are as shown in Figure 6.3.2.

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. 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 alternates⁴ (see Figure 6.3.3) chosen using the algorithm described, the highest utility bid would still be available, with two potential backups, to maximize utility under the original function. In addition, under the new utility function one of the alternate bids actually becomes the high utility bid (see Figure 6.4.3). (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 in a stable environment $k = 1$ is sufficient; this suggests that additional computational resources will be more usefully deployed generating alternatives in the case of a dynamic environment.

Since Kruskal's computes the distance between each pair of bids and then sorts the pairs, for n bids it is order $O(n^2 \log(n))$. To prevent the size from getting out of hand, if the number of bid combinations being considered exceeds one hundred, MADSUM selects a sample of 100 bid combinations from which to choose the alternatives. This keeps the alternative selection process under one second per agent as implemented (see 6.7).

⁴ These are the system defaults.

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

1. Bids from child A:

$$\begin{aligned} length_{expected} = 40, DS_{r_{expected}} &= [[3\ 5\ 2][0\ 0\ 0]], DS_{v_{expected}} = [[0\ 0\ 0][0\ 0\ 0]] \\ length_{expected} = 25, DS_{r_{expected}} &= [[2\ 2\ 1][0\ 0\ 0]], DS_{v_{expected}} = [[0\ 0\ 0][0\ 0\ 0]] \\ length_{expected} = 15, DS_{r_{expected}} &= [[3\ 3\ 1][0\ 0\ 0]], DS_{v_{expected}} = [[0\ 0\ 0][0\ 0\ 0]] \end{aligned}$$

Bids from child B:

$$\begin{aligned} length_{expected} = 40, DS_{r_{expected}} &= [[0\ 0\ 0][0\ 0\ 0]], DS_{v_{expected}} = [[3\ 5\ 2][0\ 0\ 0]] \\ length_{expected} = 25, DS_{r_{expected}} &= [[0\ 0\ 0][0\ 0\ 0]], DS_{v_{expected}} = [[2\ 2\ 1][0\ 0\ 0]] \\ length_{expected} = 15, DS_{r_{expected}} &= [[0\ 0\ 0][0\ 0\ 0]], DS_{v_{expected}} = [[3\ 3\ 1][0\ 0\ 0]] \end{aligned}$$

2. Six highest utility bids representing combinations of children of A and B:

$$\begin{aligned} length_{expected} = 80, DS_{v_{expected}} &= [[3\ 5\ 2][0\ 0\ 0]], DS_{r_{expected}} = [[3\ 5\ 2][0\ 0\ 0]] \quad U = 59.1 \\ length_{expected} = 65, DS_{v_{expected}} &= [[2\ 2\ 1][0\ 0\ 0]], DS_{r_{expected}} = [[3\ 5\ 2][0\ 0\ 0]] \quad U = 54.6 \\ length_{expected} = 65, DS_{v_{expected}} &= [[3\ 5\ 2][0\ 0\ 0]], DS_{r_{expected}} = [[2\ 2\ 1][0\ 0\ 0]] \quad U = 54.6 \\ length_{expected} = 55, DS_{v_{expected}} &= [[3\ 5\ 2][0\ 0\ 0]], DS_{r_{expected}} = [[3\ 3\ 1][0\ 0\ 0]] \quad U = 45.4 \\ length_{expected} = 55, DS_{v_{expected}} &= [[3\ 3\ 1][0\ 0\ 0]], DS_{r_{expected}} = [[3\ 5\ 2][0\ 0\ 0]] \quad U = 45.4 \\ length_{expected} = 50, DS_{v_{expected}} &= [[2\ 2\ 1][0\ 0\ 0]], DS_{r_{expected}} = [[2\ 2\ 1][0\ 0\ 0]] \quad U = 25.3 \end{aligned}$$

3. Three selected alternative Bids representing combinations of children of A and B:

$$\begin{aligned} length_{expected} = 30, DS_{r_{expected}} &= [[3\ 3\ 1][0\ 0\ 0]], DS_{v_{expected}} = [[3\ 3\ 1][0\ 0\ 0]] \quad U = 20.5 \\ length_{expected} = 15, DS_{r_{expected}} &= [[0\ 0\ 0][0\ 0\ 0]], DS_{v_{expected}} = [[3\ 3\ 1][0\ 0\ 0]] \quad U = 10.0 \\ length_{expected} = 15, DS_{r_{expected}} &= [[3\ 3\ 1][0\ 0\ 0]], DS_{v_{expected}} = [[0\ 0\ 0][0\ 0\ 0]] \quad U = 10.0 \end{aligned}$$

Figure 6.3: Bids and dissimilar alternatives and their utility under the given function.

6.4.3 Allocating Resources under Hard Constraints

Users are permitted to set constraints on the consumption of resources such as length, cost, or time. 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 6.4.4.

If the user does specify a hard constraint for a resource (such as an absolute limit of 100 words on length) then the system must separately allocate length across the agents whose bids were accepted. This avoids the possibility of peer agents in one level of the hierarchy each deciding to exceed the length specified in their

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

$$\begin{aligned} DSv_{expected} &= [[3\ 5\ 2][0\ 0\ 0]], length_{expected} = 80, DSr_{expected} = [[3\ 5\ 2][0\ 0\ 0]] \quad \mathbf{U=33.3} \\ DSv_{expected} &= [[3\ 5\ 2][0\ 0\ 0]], length_{expected} = 65, DSr_{expected} = [[2\ 2\ 1][0\ 0\ 0]] \quad U = 23.3 \\ DSv_{expected} &= [[2\ 2\ 1][0\ 0\ 0]], length_{expected} = 65, DSr_{expected} = [[3\ 5\ 2][0\ 0\ 0]] \quad U = 23.3 \\ DSv_{expected} &= [[3\ 5\ 2][0\ 0\ 0]], length_{expected} = 55, DSr_{expected} = [[3\ 3\ 1][0\ 0\ 0]] \quad U = 27.2 \\ DSv_{expected} &= [[3\ 3\ 1][0\ 0\ 0]], length_{expected} = 55, DSr_{expected} = [[3\ 5\ 2][0\ 0\ 0]] \quad U = 27.2 \\ DSv_{expected} &= [[2\ 2\ 1][0\ 0\ 0]], length_{expected} = 50, DSr_{expected} = [[2\ 2\ 1][0\ 0\ 0]] \quad U = 16.6 \end{aligned}$$

2. Three previously selected alternative Bids under new utility function:

$$\begin{aligned} length_{expected} &= 30, DSr_{expected} = [[3\ 3\ 1][0\ 0\ 0]], DSv_{expected} = [[3\ 3\ 1][0\ 0\ 0]] \quad \mathbf{U=45.8} \\ length_{expected} &= 15, DSr_{expected} = [[0\ 0\ 0][0\ 0\ 0]], DSv_{expected} = [[3\ 3\ 1][0\ 0\ 0]] \quad U = 10.5 \\ length_{expected} &= 15, DSr_{expected} = [[3\ 3\ 1][0\ 0\ 0]], DSv_{expected} = [[0\ 0\ 0][0\ 0\ 0]] \quad U = 10.5 \end{aligned}$$

Figure 6.4: Three responses, derived from different soft constraints and priority settings.

original bid, thus 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).

6.4.4 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 (see 6.4.4) and an associated expectation of utility that will be produced.

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

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 6.4.3). This MADSUM design decision is consistent with the DECAF strategy of committing to objectives, not to the plans that achieve them. Committing to objectives allows agents to achieve the objectives with the strategy that best utilizes the environment at runtime. For example, a contractor should be able to commit to building a house for a certain price, and by a certain date, without having to specify where materials are going to be purchased or exactly which

subcontractors will be used. 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. The purpose of an envelope is to allow some flexibility in accepting responses that are not exactly as negotiated. This strategy reflects the real costs, in agent time, computation, and communication, associated with rejecting a response and trying to find an alternative. The default envelope minimum is 95 percent of the utility offered by the accepted bid. If an agent has bid an alternative in addition to its accepted bid, then the cost of rejecting the result of the accepted bid is lowered, since finding that alternative is trivial. Thus when an agent has an alternative bid whose utility is higher than 95 percent of the accepted bid, the alternative bid's utility becomes the minimum acceptable utility.

To clarify the envelope concept, consider a case where the only attribute of the result that is changing is the cost, 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 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).

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 costly failure processing. Clearly the setting could depend on the domain, total resource cost and availability, user preferences, and might even need to vary across attributes. This coarse implementation of the envelope concept could be improved upon, but 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 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.

One problem faced when allowing independent branches of an agent tree to make commitments is that two branches may both commit to the maximum consumption of some resource. For example, in the text planning domain every source agent could submit a bid that would produce the length of the entire desired result. This would leave agents above with the option of allowing only a single source agent to contribute, or forcing all bidders to re-bid; and in the latter case, nothing would stop the same problem from occurring again, and being repeated at

each level of the tree[YH95]. The MADSUM architecture addresses this issue by allowing agents to make sub-optimal allocations, but employing strategies to avoid this; and by focusing on recovering from such failures gracefully (see Section 6.6).

6.4.5 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 a 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.

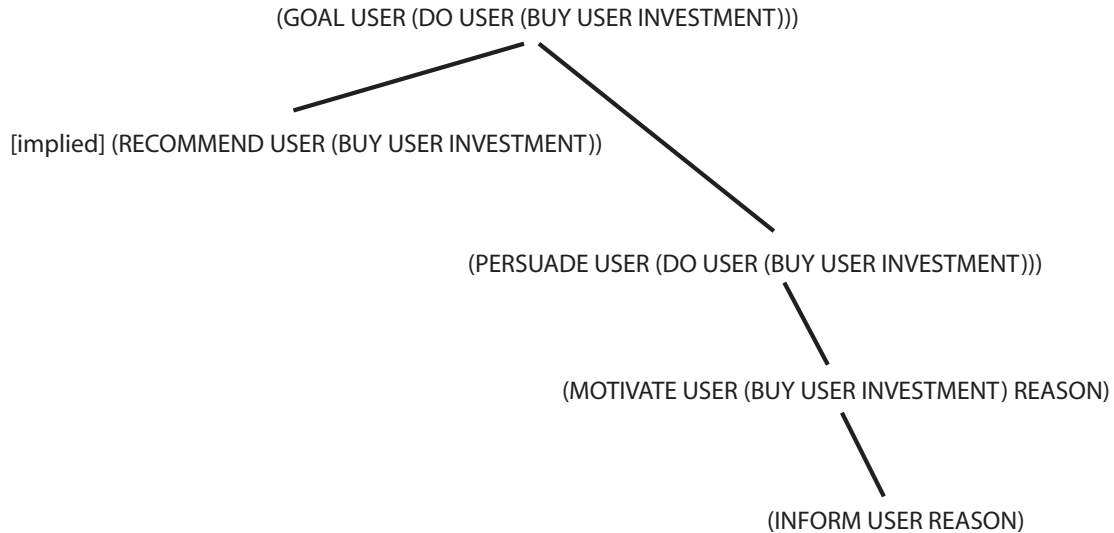


Figure 6.5: A simple text plan tree template.

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 6.5).

The text plan trees are then propagated up the agent hierarchy. The task agents above 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 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 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 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 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

MADSUM is intended to operate in a dynamic environment. While agents can predict the attributes of a result that they will provide, the attributes may in fact be very different, and as a result the utility may be different as well. The four possible combinations of possible utility and attribute⁵ results are shown in Figure 6.6. The simple cases are those numbered 2 and 4, when utility is not similar

⁵ Attributes are referred to as a class here, but this diagram could be applied to any single attribute and its relation to utility.

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.

		Utility: Results vs. Prediction	
		similar	different
Attributes: Actual vs. Predicted	similar	<p>1.</p> <p>Results are acceptable even though they may differ slightly from expectations.</p>	<p>2.</p> <p>Even though attributes are similar, the overall utility may not be acceptable. Report failure with result.</p>
	different	<p>3.</p> <p>Utility is similar even though attributes are not as expected. Assume for now that result will suffice.</p>	<p>4.</p> <p>Utility not acceptable, report failure with result.</p>

Figure 6.6: Categorizing results that vary from expectations.

6.6.1 The failure management protocol

Figure 6.7 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).

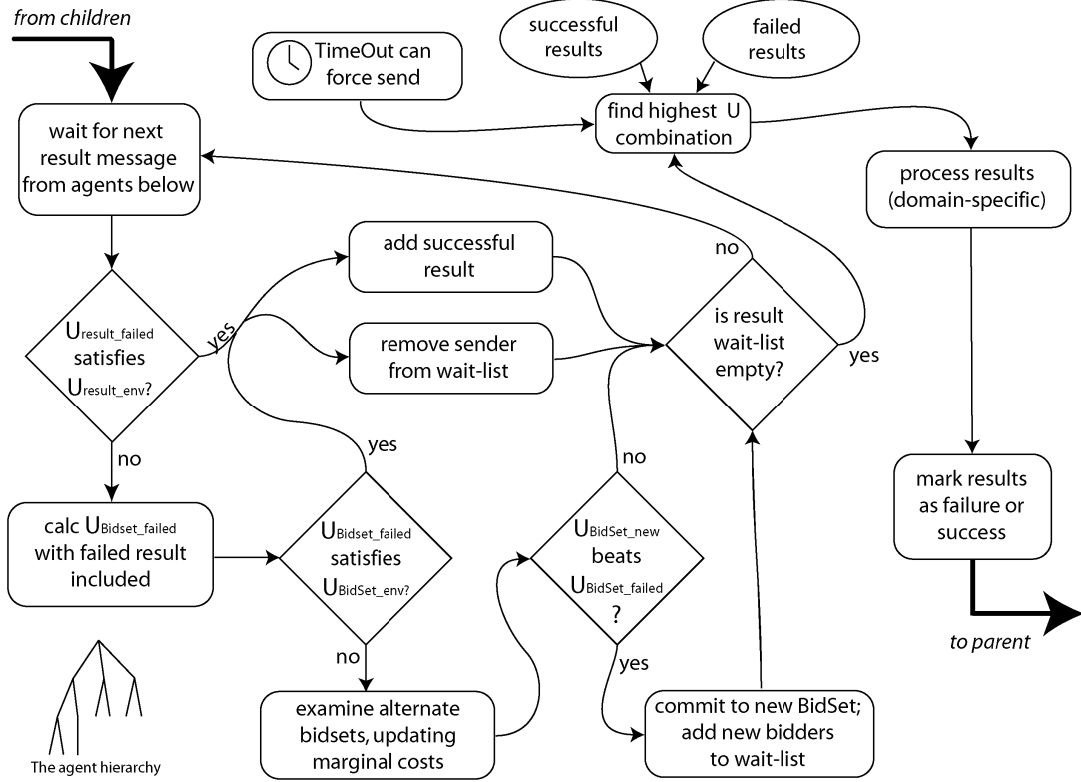


Figure 6.7: The MADSUM protocol for handling failed results.

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

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

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

Note that this marginal cost accounting greatly favors alternatives that include parts that have already been “paid for”. Furthermore, because each 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.

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

The domain processing happens last since it is (potentially) expensive relative to the attribute-based utility calculation. Simple attribute calculations are accurate in many cases (e.g. SUM works correctly for combining the weight attributes of two objects) but may only be a heuristic in others. For example, either SUM or

MAX might be correct for combining two height attributes if the relevant objects are either stacked or placed next to one another, respectively; but it is also possible that a complex 3-D rendering would be necessary (e.g. if they were stacked but partially nesting, like a cup sitting on another cup). Delaying this domain specific calculation until a set of sub-results has been chosen heuristically could possibly miss an optimal solution to a task, but also keeps computation costs low.

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 messages 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 options:

1. choose an alternate set of bids $BidSet_{alt}$ from children and communicate with children about it; or
2. decide that A has failed and report a failure to A 's parent; or
3. stay the current course and assume that the remaining utility derived from all results of $BidSet_1$ will be sufficient.

If the number of alternative sets of bids is s_{alts} , then the total number of options is $2 + s_{alts}$.

Now consider what happens when a result from a child is unsatisfactory or missing, i.e. it fails, under each possible course of action. If the agent decides to stay

the course, then it could also do so once for each other bid in the current $BidSet$, for a maximum of $|BidSet_1|$ or n_1 times, resulting in the same number of messages originally expected.

If the agent chooses to pursue 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 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 messages are still expected, and additionally n_{alt} commitment messages could be sent and n_{alt} result messages expected. However, the protocol favors choosing bid sets that overlap (due to lower marginal cost), and so it will not be n_{alt} messages 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:

$$n_{marginal} = |BidSet_{alt}| - |BidSet_1 \cap BidSet_{alt}|$$

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

In the worst case, with all results failing A will interleave choices 3 and 1, i.e. choose to stay the course for every bid in the current bid set until last bid fails (incurring a time penalty for waiting for results, but no additional communication), then choose an alternate bid set and repeat (incurring a communication cost the size of each bid set). If this case were to occur, and bid sets had no intersection with one another, the protocol would have worst case communication complexity of

$$\sum_i |BidSet_i|$$

for all alternative bidsets.

However, this worst case is almost inconceivable in practice, since MADSUM's design would be hard pressed to achieve such strange circumstances. First, the bid

sets an agent presents to a parent are very likely to largely overlap in their component bids; in fact, this is the reason for alternate bids being generated (see Section 6.4.2.1). So will the alternate bids then require large marginal communication costs if the failure mechanism gets to the point where the alternates are all that remains to try? Potentially, except that the alternate bids are designed to only be attractive if the utility function changes, so it is highly unlikely that an alternate bid would meet expected utility under the original utility function.

A more likely failure scenario, if large numbers of child result failures are occurring, is that A will run out of time and send a TIMEOUT message⁶, 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_1bid_a , C_2bid_a , and C_3bid_a , then A 's next highest utility bid set B_{alt} , might contain C_1bid_b , C_2bid_a , and C_3bid_a . Thus the marginal communication cost of changing to B_{alt} will only require one new message to C_1 .

Since the marginal communication cost of changing bid sets is typically one, the cost to A of all of A 's children failing is 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 the number of bid sets is relatively small (the system default is five high utility bid sets and 3 alternate utility bid sets), a failure response that is linear for the number of bid sets is very fast.

⁶ 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.

Still, this complexity assumes that all children are failing to meet expected utility. A more expected circumstance will be several agents each experiencing some children with some failed results. If an agent had a single child report failure, then the expected marginal communication would be one. Therefore if multiple agents had single child 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

I evaluated the performance of the architecture in three major directions. First I wanted to demonstrate that the system takes advantage of operating in parallel by distributing work across agents on different machines and so reducing overall execution time. Second, I wanted to demonstrate 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 only a linear penalty. Third, I wanted to perform controlled tests of the failure protocol to see if it would behave as predicted in Section 6.6.2.

Equation 6.1 notes the relation between the number of children of an agent and the complexity of the bid analysis. The graph in Figure 6.8 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.

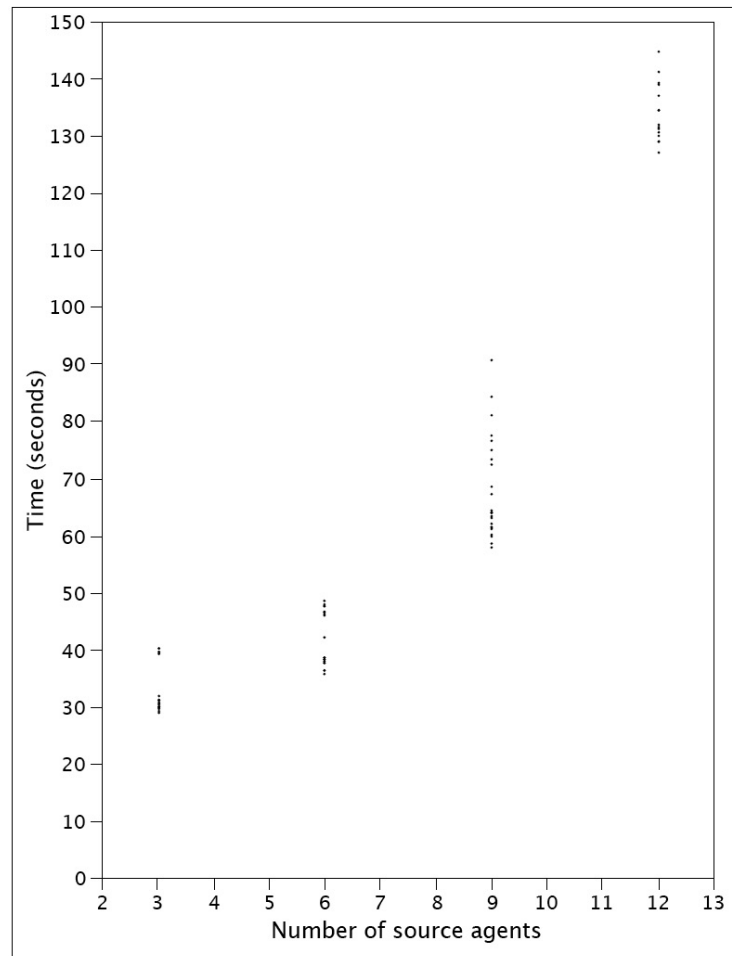


Figure 6.8: Execution time vs. Number of sources, on three machines

This data should fit Equation 6.1. More specifically, in this case 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

$$(8 + 1)^n - 1 \tag{6.2}$$

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 6.9 shows the actual data correlated with the predicted result from Equation 6.2. 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. While this strong correlation corroborates my prediction about the behavior of the system, Figure 6.8 also visually demonstrates the necessity of using the hierarchical nature of the MADSUM architecture to reduce the number of children of any given agent to avoid the exponential complexity of selecting bid combinations⁷.

Is this kind of growth problematic? Figure 6.8 shows trees of degree four performing in time under three minutes. MADSUM's current implementation has very fast, simple source agents, and so this time appears as a significant factor in total execution time. However, if source agents were more intelligent, then this degree of MADSUM overhead would not be overly burdensome. For example, the complex decision support agent BIG [LHK⁺00] reports run times in excess of forty minutes.

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

⁷ The alternative, of course, is to use additional AI techniques to select the top candidate bid combinations, rather than examining all possible combinations. While both feasible and interesting, that work remains for a future implementation of MADSUM. One clear win for examining high utility bids would be to evaluate bid combinations as they are generated to avoid exceeding memory limits when storing all combinations; however this would be more difficult for collecting alternate utility bids.

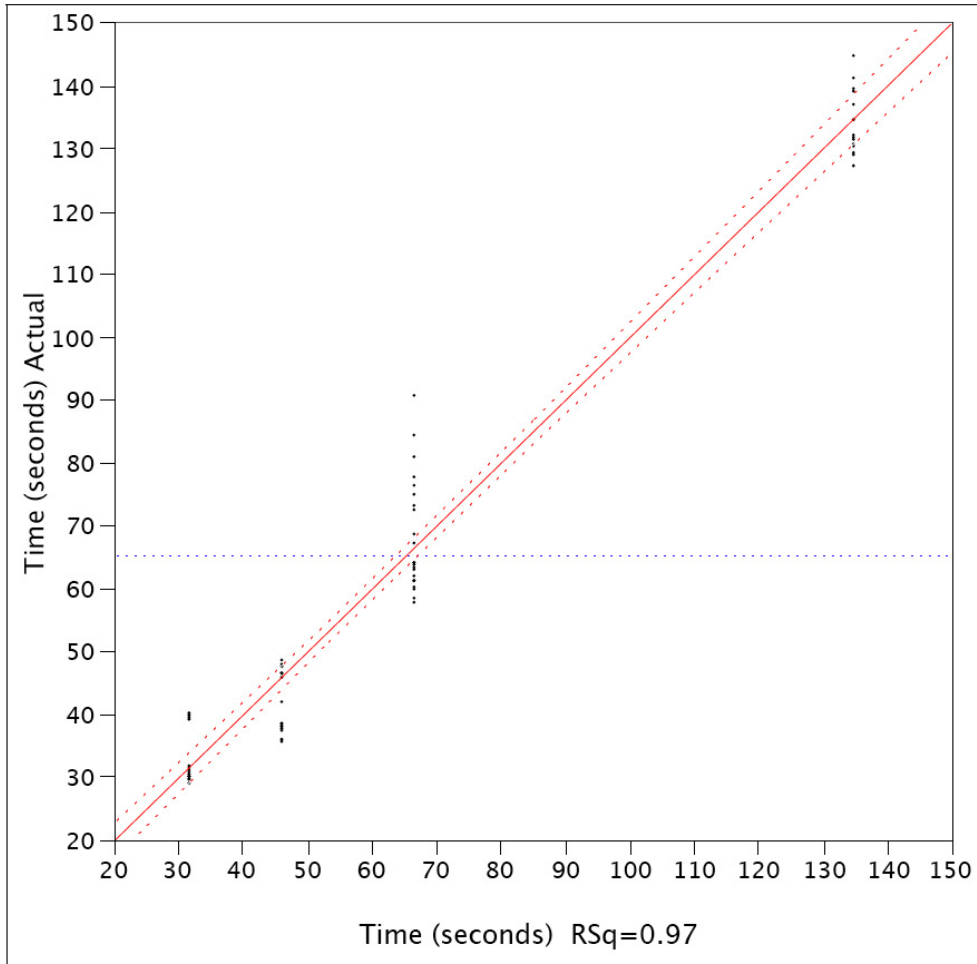


Figure 6.9: Execution time vs. Predicted Time

with the data shown in Figure 6.8; however, MADSUM scales well as long as the degree of the hierarchy is preserved. To emphasize the significance of not increasing the degree of the agent hierarchy, Figure 6.10 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. For this experiment 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, but the number of sources varies from zero to seven.

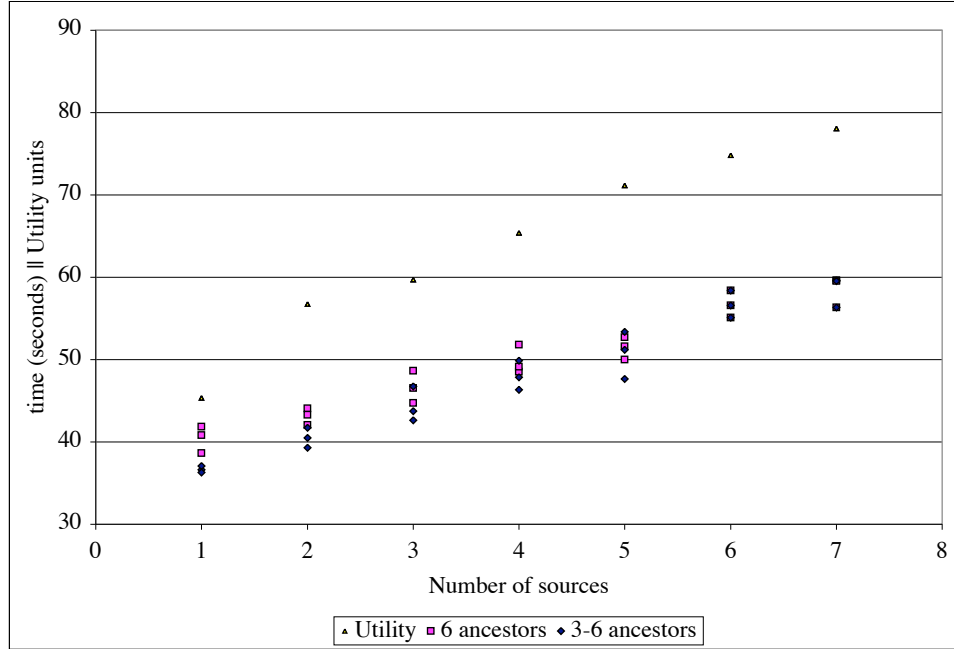


Figure 6.10: Runtime vs. sources count, 42 trials

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 extract 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.

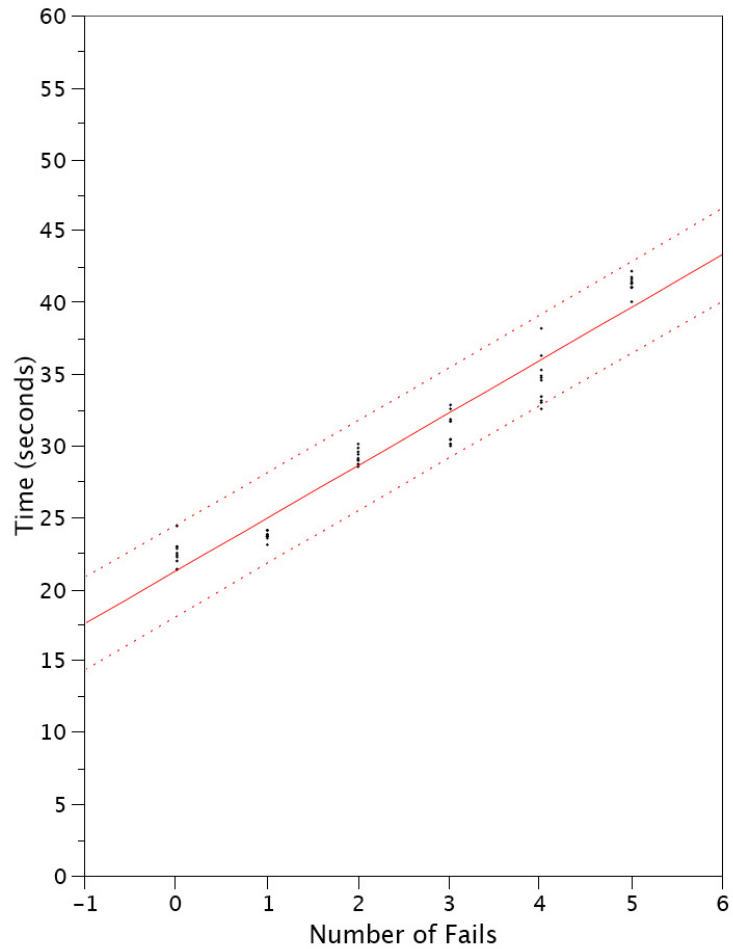


Figure 6.11: Number of agents sending one failed result each vs. Execution Time. Linear fit shown with R-square of 0.94. Dotted lines are .95 confidence boundaries.

Figure 6.11 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 agents ran on a single 867MHz laptop with 512 Mb RAM). The experiment consisted of 60 trials with the thirteen agents previously described, plus one reporting agent. Of the seven source agents, five⁸ were given data that would

⁸ Two sources under GOALwere not selected to fail since they calculate initial bids dynamically from portfolio data, so it is much harder to stage a recoverable failure.

result in the source returning a failed result. Of the five, 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 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. 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).

Figure 6.12 shows the same experiment as Figure 6.11, but this time the agents are distributed across three machines. 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 6.13 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).

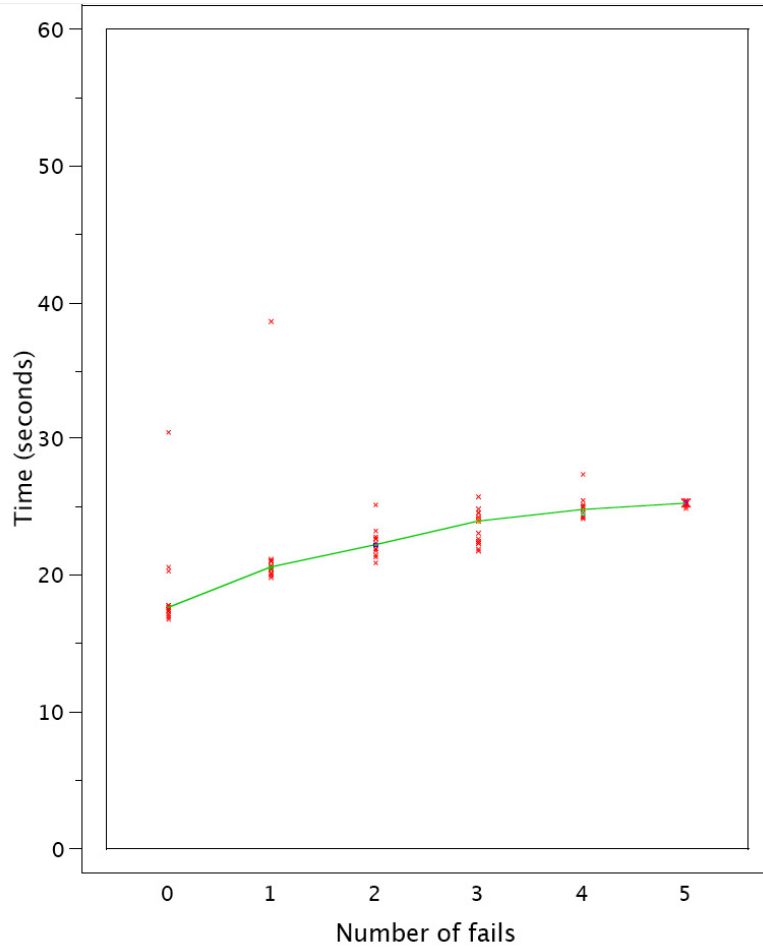


Figure 6.12: Number of agents sending one failed result each vs. execution time, with agents distributed across three machines. Line connects median values for each data group.

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 bids from each agent allow the establishment of a target utility at a high level, while MADSUM agents allocate expected utility recursively to sub-agents, allowing the sub-agents to determine precisely how to derive

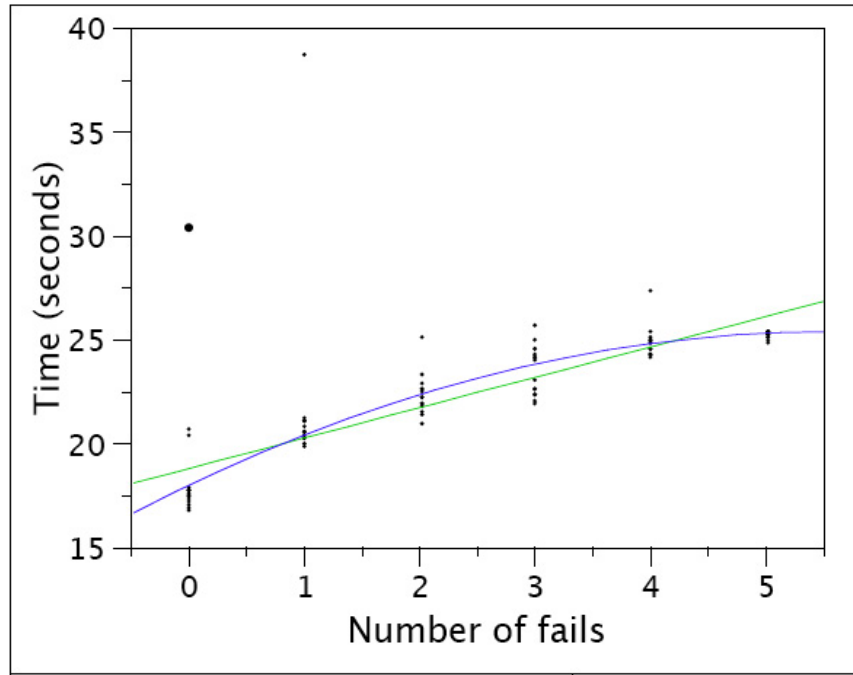


Figure 6.13: Number of agents sending one failed result each vs. Execution Time. Linear fit shown with R-Square of 0.85; 2nd degree polynomial fit shown with R-square 0.90.

that utility themselves. This mechanism provides agents with flexibility to operate in a dynamic environment. Changes in the environment of 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.

Evaluations bear out my predictions about MADSUM's performance. In particular, while increasing the degree of the agent hierarchy tree is exponentially expensive, testing shows that 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.

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.
- [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.

- [CLR92] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*, chapter 24, pages 504–505. MIT Press, 1992.
- [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.
- [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 Czakowski, 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.
- [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 assurance 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.
- [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.
- [LHK⁺00] Victor Lesser, Bryan Horling, Frank Klassner, Anita Raja, Thomas Wagner, and Shelley XQ. Zhang. Big: an agent for resource-bounded information gathering and decision making. *Artif. Intell.*, 118(1-2):197–244, 2000.
- [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.
- [McK85] Kathleen R. McKeown. *Text Generation*. Cambridge University Press, Cambridge, New York, 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.
- [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.
- [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.
- [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. In P. Ciancarini and M. Wooldridge, editors, *Agent-Oriented Software Engineering*, chapter Representing Agent Interaction Protocols in UML, 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.
- [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.
- [Ric79] Elaine Rich. User modeling via stereotypes. In *Cognitive Science*, volume 3, pages 329–354, 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.
- [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, 2002.
- [SZL04] Jiaying Shen, Xiaoqin Zhang, and Victor Lesser. Degree of local co-operation 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.
- [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. Akba: 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.
- [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.